CheckSight user manual

Documentation for Leutron Vision smart PC-based cameras

Leutron Vision

CheckSight user manual: Documentation for Leutron Vision smart PC-based cameras

Leutron Vision

Revision: 1.0-21557. Leutron Vision documentation set.

Publication date 27 October 2013
Copyright © 1995-2011 Leutron Vision

All Information in this document is subject to change without notice and does not represent a commitment on the part of Leutron Vision. The software products described in this document are furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of agreement.

It is against the law to copy the software on any medium except as specifically allowed in the license or nondisclosure agreement. The licensee may make one copy of the software for backup purposes. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or information storage and retrieval systems, for any purpose other than the licensee's personal use, without the express written permission of Leutron Vision.

Product names mentioned in this manual may be trademarks or registered trademarks of their respective companies and are hereby acknowledged.

Table of Contents

General information 8
Scope of the manual 8
Related documents 8
Where do I find…? 9
Important information 10
Precautions 10
Licensing information 10
1. Quick start 12
1.1. Prepare the camera(s) 12
1.2. Simplon installation — quick guide for Windows 12
1.3. Testing the camera(s) 14
1.4. Basic camera features 15
2. Basic information 19
2.1. Product package 19
2.2. CheckSight model naming convention 19
2.3. Ordering option summary 20
2.4. Accessories 20
2.5. Standard conformity 22
2.5.1. GenICam 22
2.5.2. CE & FCC 23
2.5.2.1. Declaration of conformity 23
2.5.2.2. Regulatory notices 23
2.5.3. RoHS 23
2.6. Environmental requirements 23
2.7. Technical specifications overview 24
3. Hardware interface and installation 25
3.1. Appearance overview 25
3.2. System architecture 26
3.3. Dimensions 27
3.4. Mounting points 29
3.5. Lens mount, optical head 30
3.6. Sensor 30
3.6.1. Sensor types 30
3.6.1.1. Progressive scanning 30
3.6.1.2. CMOS vs. CCD 30
3.6.1.3. CMOS: global vs. rolling shutter 30
3.6.1.4. CCD readout types 31
3.7. IR-cut (or other) filter 31
3.7.1. Changing or removing the filter 31
3.8. Input and output signals 32
3.8.1. DC power input 32
3.8.2. PC interfaces 32
3.8.3. Trigger input 33
3.8.4. Optocoupler interface 33
3.8.4.1. Optocoupler basics 33
3.8.4.2. Optocoupler inputs 33
3.8.4.3. Optocoupler outputs 35
3.8.5. TTL I/O's 36
3.8.6. Serial port interface 37
3.9. Connector and cable description 37
3.9.1. Power input connector 37
3.9.2. I/O connector 37
3.9.2.1. SDR-26 connector adapter cable 38
3.9.3. Ethernet connector 39
3.9.4. USB connectors 40

3.9.5. Data storage device connectors 40
3.9.6. Display connector 40
3.9.7. Audio connector 41
3.10. Status LED 41
4. Software installation 43
4.1. Operating system 43
4.2. Windows 43
4.2.1. Chipset drivers 43
4.2.2. Display driver 44
4.2.3. Network driver 45
4.2.4. Audio driver 47
4.2.5. Power management settings 48
4.3. Windows Embedded Standard 49
4.3.1. Requirements 49
4.3.2. Installation procedure for systems distributed as ISO files 49
4.3.3. List of distributed systems 50
4.3.4. Useful hints 50
4.4. Linux 51
4.4.1. Chipset, network and audio drivers 52
4.4.2. Display driver 52
4.4.3. Recommendations 52
4.5. Simplon 53
4.5.1. Installation in Windows 53
4.5.1.1. System requirements 53
4.5.1.2. Installation media 53
4.5.1.3. Installation procedure 53
4.5.1.4. Installation layout 58
4.5.2. Installation in Linux 59
4.5.2.1. Installation package types 59
4.5.2.2. System requirements 59
4.5.2.3. Standard installation layout 60
4.5.2.4. Notes for the most common distributions 62
5. Initial configuration and preparation 64
5.1. BIOS 64
5.1.1. BIOS specifications 64
5.1.2. BIOS Setup configuration 64
5.1.3. BIOS upgrade 73
5.2. Firmware update 74
5.3. Simplon configuration 76
5.3.1. Simplon Settings utility and the configuration file 76
5.3.2. Selected configuration options 77
5.4. Acquisition and runtime configuration 78
6. Camera features and their control 79
6.1. Common principles 79
6.2. Camera classes 80
6.2.1. Basic CheckSight/CheckSight-II models 80
6.2.2. OEM features 80
6.3. Information about the camera 80
6.3.1. Device identity and version 80
6.3.2. Runtime information 82
6.3.3. Resetting the camera 82
6.3.4. Pixel clock (sensor digitization clock) 83
6.3.5. Register streaming support 83
6.4. Transport layer control 84
6.4.1. Acquired image data size 84
6.5. Image format 85
6.5.1. Pixel format 85
6.5.2. Binning and decimation 86

Copyright © Leutron Vision

Leutron Vision

6.5.2.1. Binning/decimation on color cameras 88
6.5.3. Image size, area of interest (AOI) 88
6.5.3.1. AOI limits 90
6.5.3.2. AOI and binning/decimation 90
6.5.3.3. AOI on color cameras 91
6.5.4. Advanced AOI settings 91
6.5.4.1. Two-step AOI implementation 91
6.5.4.2. AOI modes 92
6.5.4.3. AOI parameter dependencies 98
6.5.5. Fixed vs. variable payload size 99
6.6. Acquisition and trigger control 101
6.6.1. Basic acquisition control 101
6.6.2. Exposure time 104
6.6.2.1. Automatic exposure 105
6.6.2.2. Exposure mode 106
6.6.3. Trigger modes 106
6.6.3.1. Free running mode 106
6.6.3.2. Triggered mode (accurate exposure) 107
6.6.3.3. Triggered mode (long range exposure) 107
6.6.3.4. Trigger modes of the CMOS rolling shutter cameras 108
6.6.4. Trigger configuration 109
6.6.5. Frame rate 112
6.7. Digital I/O, counters and timers 113
6.7.1. Camera I/O lines 113
6.7.1.1. Physical and logical lines 114
6.7.2. Observing I/O lines' status 115
6.7.3. Using input lines for triggering 115
6.7.4. Basic output (strobe) configuration 116
6.7.5. User output configuration 117
6.7.6. Advanced output configuration 119
6.7.7. Disconnected outputs 123
6.7.8. Use for inter-camera synchronization 123
6.7.9. Input lines debouncer 123
6.7.10. Special topics 124
6.8. Action commands control 125
6.8.1. Using action commands for triggering 126
6.8.2. Generating the action commands 127
6.9. Analog controls 127
6.9.1. Gain 127
6.9.1.1. Default gain and exposure time settings 130
6.9.2. Black level 130
6.9.2.1. Black level and gain 132
6.9.2.2. Black level and pixel format 132
6.9.3. Automatic gain and black level 132
6.10. Luminance and color handling 134
6.10.1. Look-up table (LUT) 134
6.10.1.1. LUT single access reading/writing 140
6.10.1.2. Direct and indirect LUT control (LUT modes) 142
6.10.1.3. White balance 142
6.10.2. Color transformation 144
6.10.3. Gamma correction, brightness and contrast 145
6.10.4. White balance 145
6.11. Chunk data 146
6.11.1. Enabling chunk data 147
6.11.2. Chunk data features naming convention 149
6.11.3. Chunk types 149
6.12. Asynchronous events 151
6.12.1. Enabling events 152

6.12.2. Event features naming convention 154
6.12.3. Event types 155
6.13. User sets (saving/loading camera configuration) 157
6.13.1. Features defining camera configuration 157
6.13.2. Camera's user configuration sets 158
6.13.3. Software control of configuration sets 159
6.13.4. Comparison of the two methods 159
6.14. External device control 160
6.14.1. Serial Port Control 160
6.15. Special features 163
6.15.1. Special purpose triggers 163
6.15.1.1. Image stamps reset 164
6.15.2. Watchdog timer control 165
6.16. Feature summary 167
7. Simplon GenTL Producer features 194
7.1. GenTL system module 194
7.1.1. Information about the system 195
7.1.2. Interface enumeration 196
7.2. GenTL interface module 198
7.2.1. Information about the interface 198
7.2.2. Device enumeration 200
7.3. GenTL device module 202
7.3.1. Information about the device 202
7.3.2. Stream enumeration 203
7.3.3. GigE Vision streaming control 204
7.3.4. GigE Vision communication control 205
7.3.5. Determining the GigE Vision streaming packet size 205
7.4. GenTL data stream module 206
7.4.1. Information about the stream 206
7.5. Simplon GenTL Producer feature summary 208
8. Connecting with 3rd party software packages 215
8.1. Halcon (MVTec Software GmbH) 215
8.1.1. GigE Vision interface (PicSight-GigE, PicSight-Smart) 215
8.1.2. GenTL interface (CheckSight/CheckSight-II, optionally PicSight-GigE/PicSight-Smart) 216
8.1.3. Common notes 217
8.1.4. Halcon Image Acquisition Assistant 218
8.2. Common Vision Blox (Stemmer Imaging) 222
8.2.1. GigE Vision interface (PicSight-GigE, PicSight-Smart) 222
8.2.2. GenTL interface (CheckSight/CheckSight-II, optionally PicSight-GigE/PicSight-Smart) 223
8.2.3. Getting started quickly 223
8.3. VisionPro, CVL (Cognex) 228
8.3.1. Installation 229
8.3.2. Functionality 229
8.3.2.1. Interfaces 230
8.3.2.2. Options 230
8.3.2.3. Device 230
8.3.3. Settings saved from Simplon Explorer 231
8.3.4. Features 231
8.4. OpenCV 232
9. Troubleshooting & Support 234
9.1. Getting support for Leutron Vision products 234
9.1.1. Before you contact the support team 234
9.1.2. Contact options 234
9.1.3. Gathering information about the problem 234
9.1.4. Reporting the problem 235
9.1.5. Return Material Authorization (RMA) 236

9.1.6. Customer registration 236
9.2. Troubleshooting Leutron Vision products 236
9.2.1. Troubleshooting common issues 236
9.2.2. Known problems and limitations 237
9.2.2.1. Simplon software package 237
9.2.2.2. CheckSight cameras 237
9.2.3. Simplon logging 237
9.2.3.1. Linux driver logging 237
9.2.4. Debugging symbols 238
A. Technical specifications for individual models 239
A.1. CheckSight G32B 239
A.2. CheckSight G32M 240
A.3. CheckSight G43AB 241
A.4. CheckSight G43AM 242
A.5. CheckSight P133B 244
A.6. CheckSight P141B 245
A.7. CheckSight P141M 246
A.8. CheckSight P142AB 247
A.9. CheckSight P142AM 249
A.10. CheckSight P202B 250
A.11. CheckSight P202M 251
A.12. CheckSight P203B 252
A.13. CheckSight P32B 254
A.14. CheckSight P32M 255
A.15. CheckSight P33B 256
A.16. CheckSight P33M 257
A.17. CheckSight P34B 259
A.18. CheckSight P34M 260
A.19. CheckSight P52B 261
A.20. CheckSight P52M 262
A.21. CheckSight P83B 264
A.22. CheckSight P83M 265
A.23. CheckSight R132M 266
A.24. CheckSight R312AB 267
A.25. CheckSight R502B 269
A.26. CheckSight R502M 270
Contacting Leutron Vision 272
Headquarters (Switzerland) 272
Germany 272
Other countries 272
Useful links 272

General information

Scope of the manual

This manual introduces the CheckSight camera family and covers all common aspects of work with these cameras. Technical specifications and other model-specific data are listed in an appendix (Appendix A, *Technical specifications for individual models* [p. 239]).

Related documents

- Simplon getting started — Simplon package overview, installation instructions and quick start guide.
- Product datasheets — list of Leutron Vision camera models with links to their respective datasheets.
- PicSight-GigE user manual — user manual for our GigE Vision camera series.
- Simplon tools — overview and documentation of various tools delivered with the Simplon package.

Copyright © Leutron Vision

Where do I find…?

The following points could help you to search for some important piece of information within this manual:

- Install and test the camera quickly: Chapter 1, *Quick start* [p. 12]
- Technical specifications: Appendix A, *Technical specifications for individual models* [p. 239]
- Connector pinout description: Section 3.9, “Connector and cable description” [p. 37]
- Simplon package installation instructions: Section 4.5, “Simplon” [p. 53]
- Detailed documentation of camera features: Chapter 6, *Camera features and their control* [p. 79]
- GenICam and other industrial standards: Standards's introduction in Simplon getting started
- Release notes (revision history): CheckSight revision history
- Looking for something else? Try Table of contents.

Copyright © Leutron Vision

Important information

Precautions

Not respecting these precautions can lead to injury, camera damage, incorrect camera performance and loss of warranty.

- **Read the documentation.** Read the camera documentation before using the camera.
- **Camera power.** Incorrect input power can damage the camera. Follow guidelines in Section 3.8.1, “DC power input” [p. 32]. Do not reverse power polarity. Do not connect or disconnect other cables when the camera power is on.
- **Opening the camera.** Do not open the camera. Do not let liquid, dust, flammable or metallic material to get inside the camera.
- **Environment.** Always use the camera in conditions meeting the requirements specified in Section 2.6, “Environmental requirements” [p. 23]
- **Cleaning the sensor window.** Avoid cleaning the sensor window if possible. Keep lens cap closed as long as no lens is attached, avoid touching the sensor. If necessary, clean the sensor window using compressed air. If further cleaning is required, use lint-free, ESD-safe cloth wiper. Avoid cloth that could generate static charge or that could scratch the window. The camera should be cleaned in an ESD-safe area, the person performing cleaning should be earthed.
- **Electromagnetic fields.** Keep the camera away from strong electromagnetic fields. Avoid static charging, handle the camera in ESD protected area.
- **Connectors.** Take care when handling the camera so that no damage can be done to the connectors. Prevent foreign objects in the connectors.
- **Transporting the camera.** Always transport the camera in its original packaging.

Licensing information

The Simplon package and/or the software included in the cameras is using following 3rd party software components:

- GenICam GenApi reference implementation — distributed under the GenICam license by the GenICam committee. The copyright is held by the GenICam committee and its contributing members working group (which includes Leutron Vision). The package as well as the license is available from www.genicam.org[1]. The GenICam GenApi reference implementation in turn uses Apache Xalan-C++ and Xerces-C++ libraries (Apache license, www.apache.org[2]) and the modified MathParser librariy (LGPL license, kirya.narod.ru/mathparser.html[3]).
- zlib compression library: available from www.zlib.net[4] under zlib license.
- lwIP (lightweight TCP/IP stack): available from savannah.nongnu.org/projects/lwip[5] under modified BSD license. The copyright © is held by Swedish Institute of Computer Science.

[1] http://www.genicam.org
[2] http://www.apache.org/
[3] http://kirya.narod.ru/mathparser.html
[4] http://www.zlib.net/
[5] http://savannah.nongnu.org/projects/lwip/

- The Independent JPEG Group's JPEG software (libjpeg): available from www.ijg.org[6] (Independent JPEG Group).
- OpenCV: available from opencv.willowgarage.com[7] under BSD license.

[6] http://www.ijg.org/
[7] http://opencv.willowgarage.com/

1. Quick start

This chapter is intended to help users wanting to start quickly with basic usage of the cameras, without studying the entire documentation. The information in this chapter duplicates information in the following parts of the manual. Whenever in doubt, please refer to the corresponding chapters of the manual, providing detailed description of given problems.

1.1. Prepare the camera(s)

1. Unpack the camera, connect all the necessary peripheral. If needed, check the interface connectors description in Section 3.9, “Connector and cable description” [p. 37].

2. Power up the camera using proper 10—26V power supply.

3. Assuming that the desired operating system is already pre-installed on the camera, no additional steps are needed and the camera is ready for Simplon installation ant test. Otherwise, install the operating system, following instructions of the operating system supplier. Install the necessary device drivers available from our download area[1].

1.2. Simplon installation — quick guide for Windows

Following points guide you quickly through Simplon installation under Windows. If in doubts, if you need detailed instructions or if installing in another operating system, refer to full Simplon installation description.

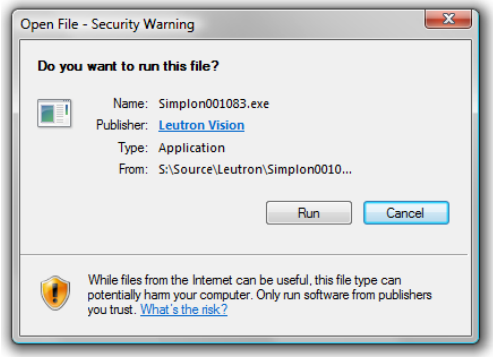
Now let's walk through the installation procedure:

1. Prepare the installation media. The software should be installed by a user with administrator rights.

   - If you have the installation CD, insert it in the CD-ROM drive and wait until installation starts.

   - If you have downloaded the installer from our website[2], execute it.

2. Depending on security adjustments of your system and whether you are running the installer from a local disk or a network share, you might get following security warning. Simply proceed clicking Run.


Open File - Security Warning
Do you want to run this file?
Name: Simplon001083.exe
Publisher: Leutron Vision
Type: Application
From: S:\Source\Leutron\Simplon0010...
Run
Cancel
While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

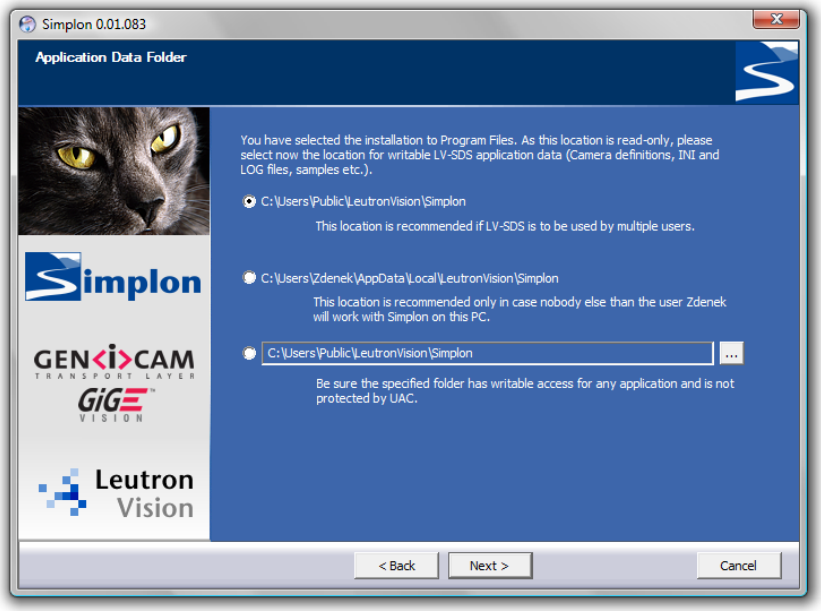
3. The installer welcomes you with following screen, click Next.

[1] http://www.leutron.com/support-downloads/download-area/
[2] http://www.leutron.com/support-downloads/download-area/

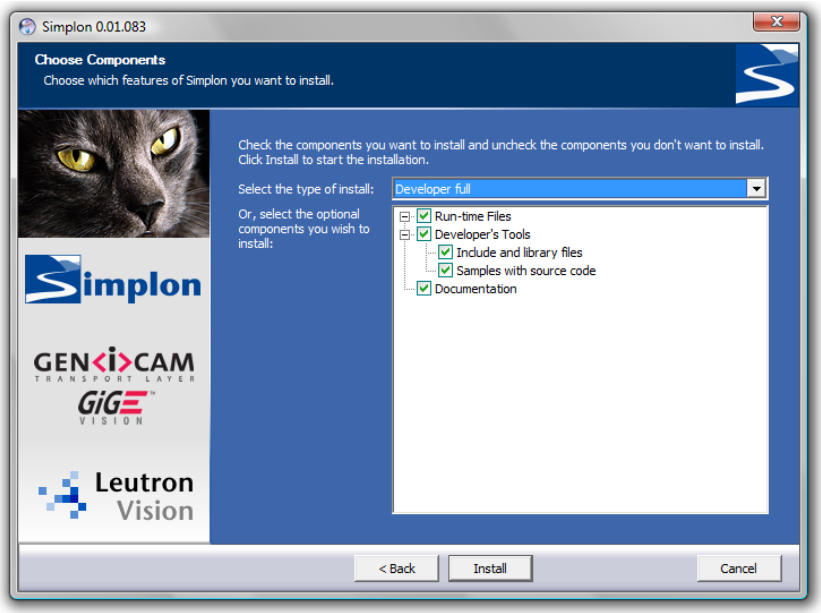
4. Next screen is the license agreement. Please read carefully the license text and if you agree, click the I Agree button to proceed with installation.

5. Next the installer asks you about the location where to install Simplon. The default location will do well in most cases. Keep it or select another preferred location and click Next.

6. If running Windows Vista or newer Windows version, the default software installation directory (`C:\Program Files`) is read only. The installer will ask you where to put the application data requiring write access (the dialog will not appear on Windows XP). Select a desired location and click Next.


Simplon 0.01.083
Application Data Folder
You have selected the installation to Program Files. As this location is read-only, please select now the location for writable LV-SDS application data (Camera definitions, INI and LOG files, samples etc.).
C:\Users\Public\LeutronVision\Simplon
This location is recommended if LV-SDS is to be used by multiple users.
C:\Users\Zdenek\AppData\Local\LeutronVision\Simplon
This location is recommended only in case nobody else than the user Zdenek will work with Simplon on this PC.
C:\Users\Public\LeutronVision\Simplon
Be sure the specified folder has writable access for any application and is not protected by UAC.
Simplon
GEN<i>CAM
GiGE
Leutron Vision
< Back
Next >
Cancel

7. In the next step you can select the Simplon components to install. Note that for testing the cameras, the “runtime” components suffice. However, for programming with Simplon you need to make sure the Developer's tools are installed as well. When finished with the selection, click Install.



8. After starting the installation, the selected components will be installed to your computer. The installer will inform you about the progress.

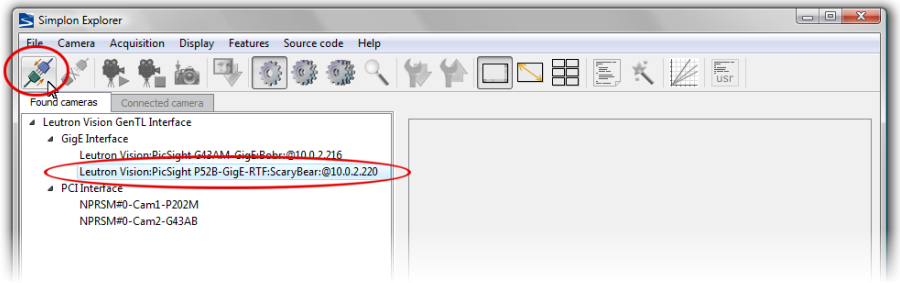
9. When completed, the installer displays the final dialog. Keep the default option (Reboot now) and click Finish. After rebooting, Simplon is ready to use.

1.3. Testing the camera(s)

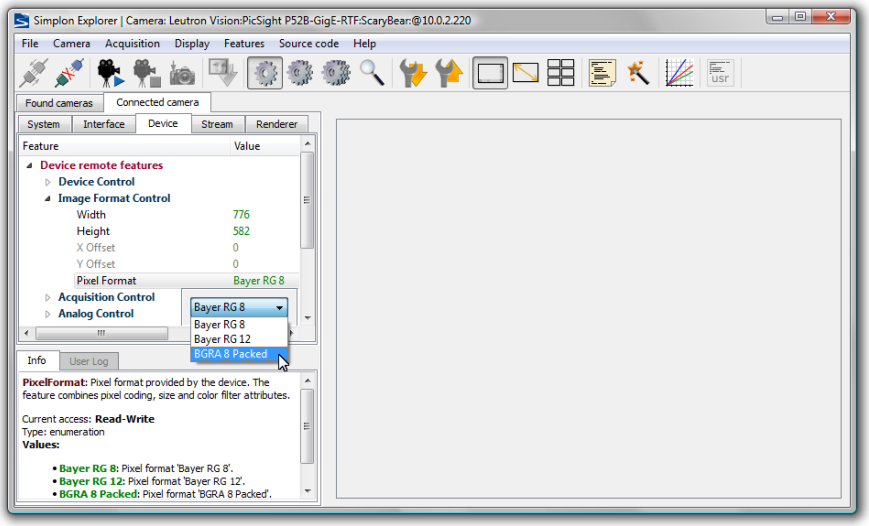
The Simplon package contains the Simplon Explorer tool, which allows to enumerate, connect and configure camera(s), acquire images or generate sample source code for Simplon API. It is a useful tool for testing both the Simplon and camera functionality.

1. Start the tool from system menu: Start → Programs → Leutron Vision Simplon → Simplon Explorer (under Linux, start `/opt/simplon/bin/lv.explorer`).

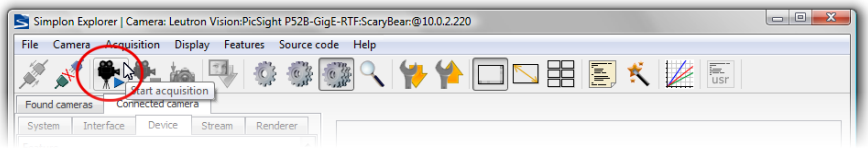
2. Connect the camera to be tested — select it in the list of found cameras and press the *Connect camera* button. Of course, the camera must be connected and powered.



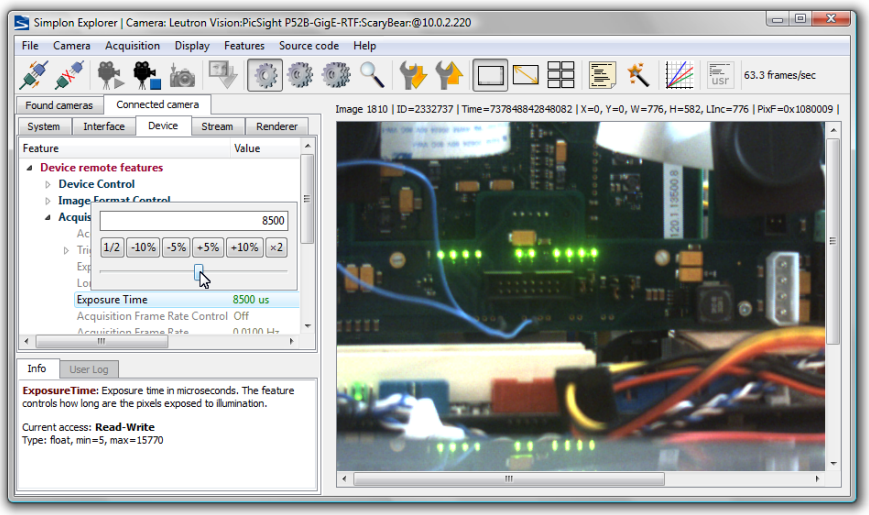
3. Configure the camera features as desired. Pay attention especially to features in `Image Format Control` and `Acquisition Control` categories.



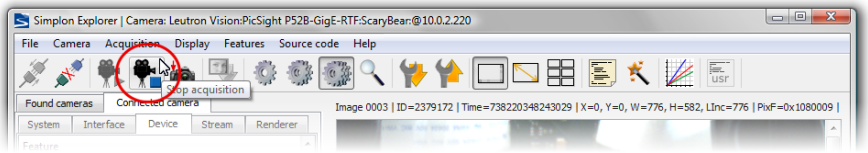
4. When prepared, click the Start acquisition button — the camera should start acquiring. If not, verify again, if the camera is properly connected and running, if the system was properly configured (remember the notes above about firewall, network configuration, etc.) or if the camera was not set to triggered mode, while no trigger was attached.


Simplon Explorer | Camera: Leutron Vision:PicSight P52B-GigE-RTF:ScaryBear:@10.0.2.220
File Camera Acquisition Display Features Source code Help
Start acquisition
Found cameras
Connected camera
System Interface Device Stream Renderer

5. While the acquisition is active, you can still adjust the runtime parameters, such as exposure time or gain. Basic acquisition parameters, such as pixel format or trigger mode become locked when the acquisition starts.



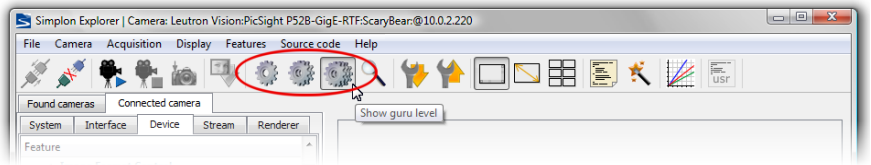
6. When finished, click the Stop acquisition button and exit.


Simplon Explorer | Camera: Leutron Vision:PicSight P52B-GigE-RTF:ScaryBear:@10.0.2.220
File Camera Acquisition Display Features Source code Help
Stop acquisition
Found cameras
Connected camera
System Interface Device Stream Renderer

1.4. Basic camera features

In this section we'll briefly discuss the most essential camera features important for the basic tests. Full description of all camera features is provided in Chapter 6, *Camera features and their control* [p. 79].

The Simplon Explorer displays by default only the “basic” camera features intended to handle the most common tasks. To display also the more advanced features, you need to switch to the “expert” feature level. Note that the third available level, “guru”, is intended just for debugging and should not be used for normal operation.


Simplon Explorer | Camera: Leutron Vision:PicSight P52B-GigE-RTF:ScaryBear:@10.0.2.220
File Camera Acquisition Display Features Source code Help
Found cameras
Connected camera
Show guru level
System Interface Device Stream Renderer

- **Image format.** The basic properties of the acquired image can be configured through features in the *Image Format* category:

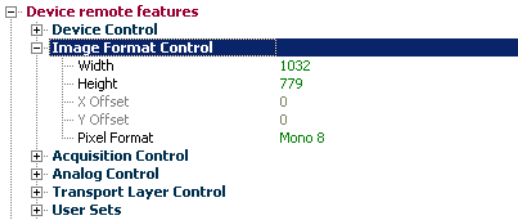
Pixel format

The *Pixel Format* feature is an enumeration allowing to specify the type of image data delivered to the application buffer. The available pixel formats differ among camera models. In general, they can be

- Monochrome, with various number of bits per pixel, eg. *Mono 8* or *Mono 12*.
- Bayer encoded, with sensor-dependent Bayer formats, eg. *Bayer GR 8* or *Bayer BG 8*. In this case the application needs to decode the Bayer encoded image buffer to get a proper color image — Simplon API provides functions for this.
- RGB color format, providing true RGB output, in this case the Bayer decoding is performed directly by the camera.

Image size

The camera will acquire full size image by default. The acquired image size can be reduced using the *Width* and *Height* features, which set the acquired image size in pixels. The field of view will be automatically centered within the full sensor image.



- **Acquisition parameters.** The luminance of the acquired image can be controlled either explicitly through exposure time and gain functions, or automatically through corresponding auto-functions.

Exposure time

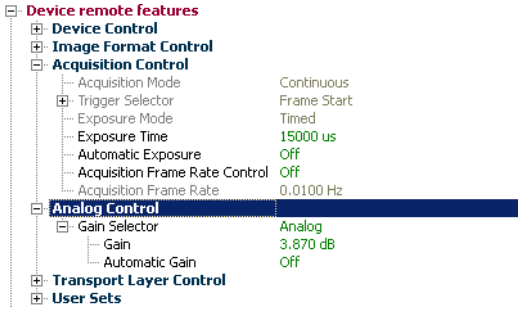
The *Exposure Time* feature in *Acquisition Control* category configures the length of the exposure in microseconds.

Analog gain

Analog gain to be applied on the sensor during acquired image digitization can be configured through corresponding features in *Analog Control* category. Verify that the *Gain Selector* feature is set to *Analog* (some cameras models support also digital gain). Now the *Gain* feature controls the desired analog gain value in decibell.

Auto-functions

Selected camera models support automatic gain and/or automatic exposure functions, when the camera keeps adjusting given parameter automatically, to maintain constant image luminance. The features will be displayed in the feature tree only if the camera supports them. To switch the auto-function on, navigate to corresponding control feature, *Automatic Exposure* in *Acquisition Control* category or *Automatic Gain* in *Analog Control* category and switch them from *Off* to *Continuous*.



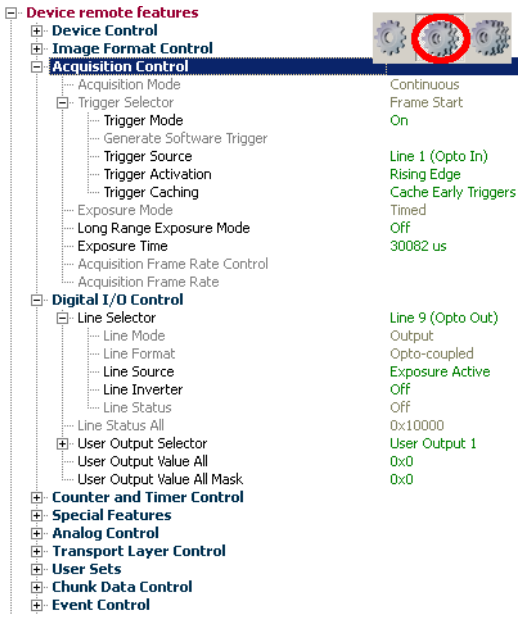
- **Triggered acquisition.** The camera starts by default in “free running” mode — acquiring continuous stream of image frames. On the other hand in the triggered mode, the acquisition of individual frames can be explicitly controlled by hardware or software triggers. The trigger related features are located in the *Acquisition Control* category.

  Triggered mode — To switch the camera to triggered mode, open the *Trigger Selector* subtree and set *Trigger Mode* to *On*. After starting the acquisition again, the camera will acquire images only when explicit trigger signal arrives.

  Trigger source — The *Trigger Source* enumeration allows to select where should the camera “listen for triggers. You can select either *Software* or one of the camera's I/O lines. The assignment of the I/O lines to the connector pins is described in Section 3.9, “Connector and cable description” [p. 37].”

  Software triggering — The software triggers can be issued during active acquisition in triggered mode, regardless whether the trigger source is set to *Software* or a physical I/O line. To apply the software trigger, select the *Generate Software Trigger* feature and click the button which pops up.

  Strobe — The strobe can be configured using features in the *Digital I/O Control* category, after switching the feature visibility level to “expert”. Select the desired output line, where the strobe pulses should be applied through *Line Selector* (the I/O line mapping to interface connector pins is described in Section 3.9, “Connector and cable description” [p. 37]). Switch the strobe on for the selected line by setting *Line Source* to *Exposure Active*. The camera will issue a strobe pulse aligned with the exposure for every acquired frame. The strobe output is available only in triggered mode.



- **Information about the camera.** It is frequently necessary to query the basic information about the camera. The information is available through features in the *Device Control* category.

  Device identity — The most important features identifying the camera are *Vendor Name* and *Model Name*. The camera's serial number is stored as string in feature *Device ID* (available only in “expert” feature level). Finally, the “nick-

name" used to identify the device is accessible (read/write) in *Device User ID*.

Firmware version

The version of the firmware loaded on the camera is accessible as string in feature *Device Firmware Version*.

2. Basic information

2.1. Product package

The standard product package contains just the camera itself.

Any required camera accessories (Section 2.4, “Accessories” [p. 20] should be ordered separately.

Camera control software (Simplon), device drivers and other updates can be downloaded from our download area[1] free of charge.

2.2. CheckSight model naming convention

To better understand CheckSight model names, we outline below the basic rules used for naming CheckSight cameras. This reference should help you understand the camera names and the available camera options. Note that this explanation is just a hint, the naming convention can be changed any time in future.

Every CheckSight name has following form: **`Annnm(Rs)C(F)`**. Meaning of individual name components is explained below:

`A` Sensor type. Possible values are:
- `P` — CCD sensor, progressive scan, interline transfer
- `G` — CMOS sensor with global shutter
- `R` — CMOS sensor with rolling shutter
- `T` — CCD sensor, progressive scan, frame transfer
- `F` — CCD sensor, progressive scan, frame readout
- `I` — CCD sensor, interlaced (2 or multiple fields)

`nnn` Total number of pixels divided by 100000 and rounded. Leading zeros are not displayed (which results in 1–3 digit number).

`m` Imager size. Possible values are:
- `0` — 1" or bigger
- `1` — 2/3" or bigger
- `2` — 1/2", 1/1.8" or bigger
- `3` — 1/3", 1/2.7" or bigger
- `4` — 1/4" or bigger
- `5` — 1/5" or bigger
- `6` — 1/6" or bigger

`Rs` Optional. Sensor revision. Allows to distinguish a new revision of the same sensor of other manufacturer or another, very similar sensor with same basic properties. The original model has the revision flag omitted. Additional revisions are flagged with `A`, `B`, etc.

`C` Color coding. Possible values are:
- `M` — monochrome
- `I` — monochrome, IR sensitive
- `B` — Bayer encoded color

`F` Optional. Optical filter. If omitted, specifies a camera with the default IR-blocking filter. Other options are:
- F — float glass filter
- D — daylight blocking filter
- N — no filter

[1] http://www.leutron.com/support-downloads/download-area/

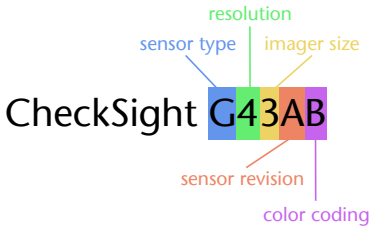


Figure 2.1. CheckSight naming convention illustrated

2.3. Ordering option summary

Every CheckSight basic model (see Appendix A, *Technical specifications for individual models* [p. 239]) can be further customized through additional options.

Following ordering options are available for high quantity OEM's:

IR-cut filter — By default, the cameras are delivered with an IR-cut filer mounted on top of the sensor. Optionally, every modelcan also be delivered without this filter. See more details in Section 3.7, "IR-cut (or other) filter" [p. 31]. Query us about possibility to deliver the camera with other (e.g. daylight blocking or float glass) filter.

Lens mount — The cameras can be delivered with CS-mount optical head instead of the standard C-mount version.

Other options might be available for high quantity OEM's, please contact Leutron Vision for more information.

2.4. Accessories

The list of CheckSight accessories might grow in future, please contact Leutron Vision to learn all the options and to get the ordering/pricing information.

CheckSight startup kit — Starter-kit consisting of 2 pcs. USB-adapter cables (32050), 1 pc. audio I/O adapter cable (32040), 1 pc. I/O-cable (32032) and 1 pc. Display cable (32012). Wide range input power supply (100-240VAC) with different wall plugs for international use (73520) as well as a CD with the Simplon software (16060). Ordering number: 32090.

Cables

- Display cable to connect a CheckSight to a DVI-D compatible display, 1.8m. Ordering number: 32012.
- Cable with SDR-26-pin female connector to connect the I/O signals of CheckSight to customized hardware, other side of the cable has mounted D-sub 25-pin female connector, 3m. Ordering number: 32032.
- Adapter cable for CheckSight to split the audio I/O connector (3.5mm male audio jack) into 2 separate 3.5mm female audio jacks for microphone input and audio output, 0.2m. Ordering number: 32040.
- USB type A male to USB type A female connector. Very tiny connectors which allow to connect all 3 USB cables, the sound cable and the display cable simultaneously (space constraints of CheckSight connector panel). With this device bigger USB-sticks can be connected, 0.2m. Ordering number: 32050.

Connectors	• Locking power plug for CheckSight power input connector. Ordering number: 39000475. • 26-pin SDR male plug with hood/solder version for CheckSight I/O connector. Ordering number: 24199.
Power supply	Power supply for CheckSight, 100-240VAC input, 12VDC/3.3A output, 2.5mm/DC-Power connector. Ordering number: 73520.
Simplon software package	CD with Simplon software suite for Windows-XP/7 and Linux, including Simplon Explorer to control PicSight and CheckSight cameras, comprehensive programming library with automatic source code generator, compatible to GenICam, GenTL and GigE Vision standards, Simplon GigE Vision Configuration Tool, firmware maintenance tool, software drivers for CheckSight and all related hardware and software documentation. Ordering number: 16060.
Storage devices	• CompactFlash, 4 GB, industrial grade, type I. Ordering number: 29010. • CompactFlash, 8 GB, industrial grade, type I. Ordering number: 29020.

Selected accessories are illustrated at the following photographs for better understanding.

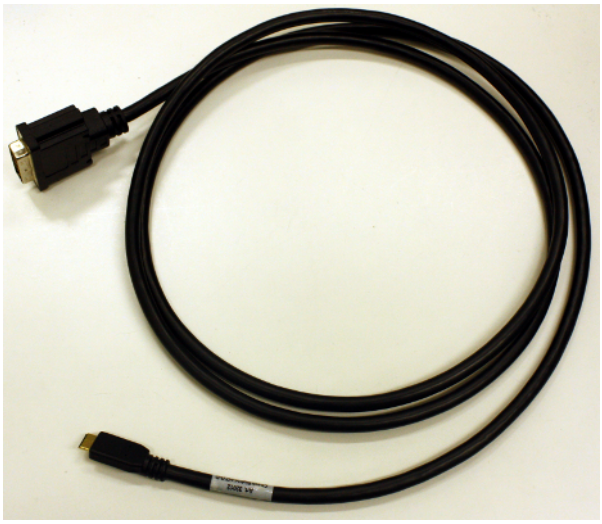


Figure 2.2. CheckSight display cable (#32012)

Figure 2.3. CheckSight I/O cable (#32032)

Figure 2.4. CheckSight audio cable (#32040)

Figure 2.5. CheckSight USB extender cable (#32050)

Figure 2.6. CheckSight locking power plug (#39000475)

2.5. Standard conformity

2.5.1. GenICam

The cameras (and supporting libraries) fully implement the GenICam standard, including SFNC (Standard Features Naming Convention) and GenTL. Thanks to the full GenTL compatibility, the cameras are plug&play compatible with all GenTL capable libraries, such as Halcon, Common Vision Blox or MATLAB.

Detailed information about these industrial standards and their use in our products is provided in Chapter 8, *Vision Standards*

2.5.2. CE & FCC

The cameras fulfill EMC standards 55022:1998/A1:2000/A2:2003 (emissions) and 55024:1998/A1:2001/A2:2003 (immunity).

2.5.2.1. Declaration of conformity

DECLARATION OF CONFORMITY
KONFORMITÄTSERKLÄRUNG

Equipment : Camera System
Geräteart

Trademark : CheckSight
Handelsmarke

Type : CheckSight
Typ

Manufacturer : Leutron Vision GmbH
Hersteller

Further Description : See technical data sheet
Weitere Angaben *Siehe Technisches Datenblatt*

The signing legal authorities state, that the above mentioned equipment meets the requirements for EMC according to
Die Unterzeichnenden erklären als rechtsverbindliche Bevollmächtigte, dass das oben erwähnte Gerät den folgenden EMV - Anforderungen entspricht

- DIRECTIVE 2004/108/ EC Electromagnetic Compatibilty (EMC)
  RICHTLINIE 2004/108/ EG *Elektromagnetische Verträglichkeit (EMV)*

The following harmonized standards have been used:
Folgende harmonisierte Normen wurden angewandt:

- EN 61000-6-4 : 2007
- IEC 61000-6-4 : 2006
- EN 61000-6-2 : 2005
- IEC 61000-6-2 : 2005

Test report EMCKP1520A

Testing Laboratory : EMC-Testcenter Zürich AG, Postfach 268, CH-8052 Zurich
accredited according to ISO / IEC 17025, Notified Body No. 1932

Name: Leutron Vision GmbH

Authority
Bevollmächtigter

Konstanz	16.02.2011	
City *Ort*	Date *Datum*	Signature *Unterschrift*

2.5.2.2. Regulatory notices

The cameras should be used with properly shielded cables (and other equipment) recommended in this manual.

Please contact us if you need further information regarding the necessary equipment.

2.5.3. RoHS

All units are RoHS compliant.

2.6. Environmental requirements

- Operating temperature: 0 °C–50 °C

- Operating relative humidity: 20 %–95 %, non-condensing
- Allow sufficient air circulation around the camera to prevent heat-up

If exposed to environment conditions outside the specified limits, camera performance can be significantly degraded or the camera can be damaged.

2.7. Technical specifications overview

Since the CheckSight family contains variety of camera models, featuring different image sensors and differing in other options, lot of the technical specifications are strongly model specific. The following table provides just a brief overview of the important common properties. The model specific information is provided in Appendix A, *Technical specifications for individual models* [p. 239]. Details about individual camera components and their properties are given in corresponding chapters of the manual, in particular in Chapter 3, *Hardware interface and installation* [p. 25].

Specification	**CheckSight camera family**
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor range	CCD & CMOS, monochrome & Bayer encoded color, progressive scanning, various trigger modes
Sensor specific info	Refer to Appendix A, *Technical specifications for individual models* [p. 239]
Camera configuration options	Refer to functional description in Chapter 6, *Camera features and their control* [p. 79]
Image acquisition & transfer	DMA to the CheckSight host memory
Lens mount	C-mount (CS-mount on request)
Dimensions	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical), 25 W (max)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, Open CV
Ordering information	Contact Leutron Vision for latest ordering and pricing information

Table 2.1. CheckSight family, common technical specifications

3. Hardware interface and installation

3.1. Appearance overview

Figure 3.1. CheckSight camera, back-side view

Figure 3.2. CheckSight camera, front-side view

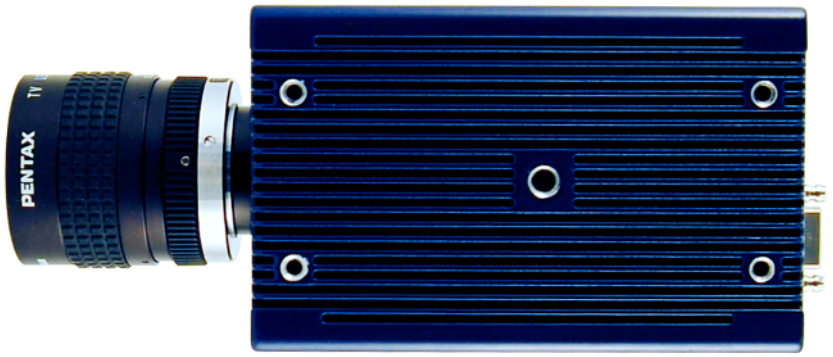


Figure 3.3. CheckSight camera, top view with mounting points

Figure 3.4. CheckSight camera, back view with connectors

3.2. System architecture

CheckSight is a complete embedded PC system built around intel ATOM Z530 processor and US15W system controller hub, both belonging to the intel embedded roadmap. Any of the wide choice of Leutron Vision's sensors may be used in CheckSight (the sensor selection is equal to that of the PicSight camera series). The PC offers LAN (10/100/1000 via RJ45), DVI-D display adapter cable and Microphone In/Speaker out as well as USB 2.0 connectors. Different boot and storage media exist.

Sensor data is acquired using an FPGA based streaming module. There, pre-processing of raw sensor data, input look-up-table, format conversion, buffering is done, before the data is transferred to the main PC using the high performance PCI-Express DMA transfer. In addition, a programmable timer unit allows control and synchronizing image acquisition to external events. General purpose I/O's (optocoupler & TTL inputs and outputs, RS232 interface) enable the control of external devices.

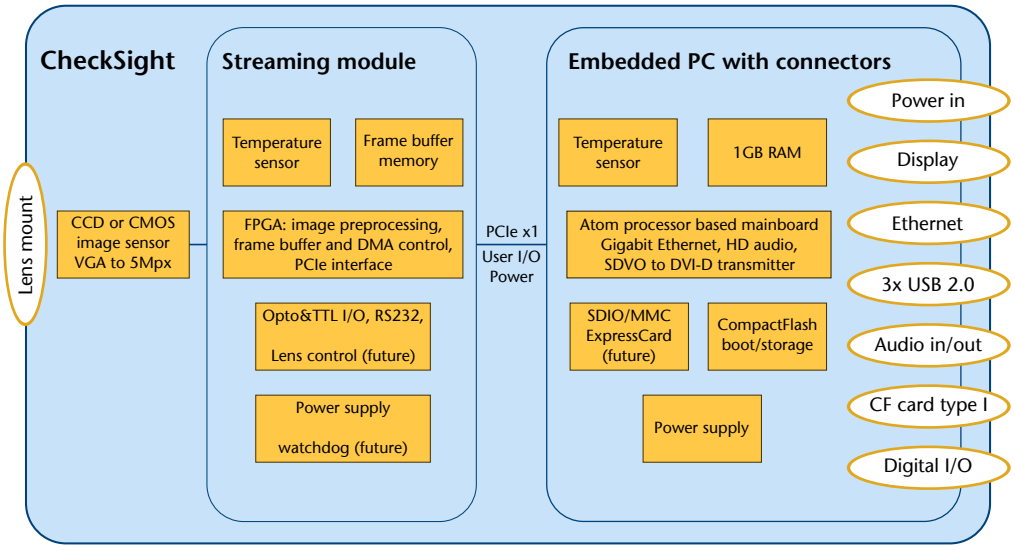


Figure 3.5. CheckSight architecture

3.3. Dimensions

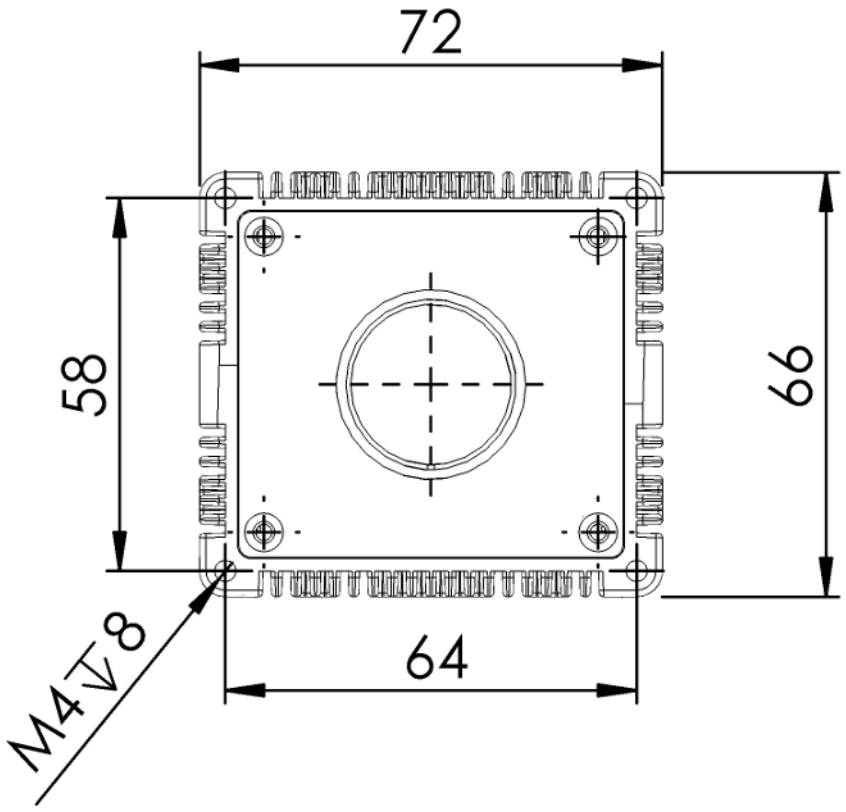


Figure 3.6. CheckSight dimensions, front view

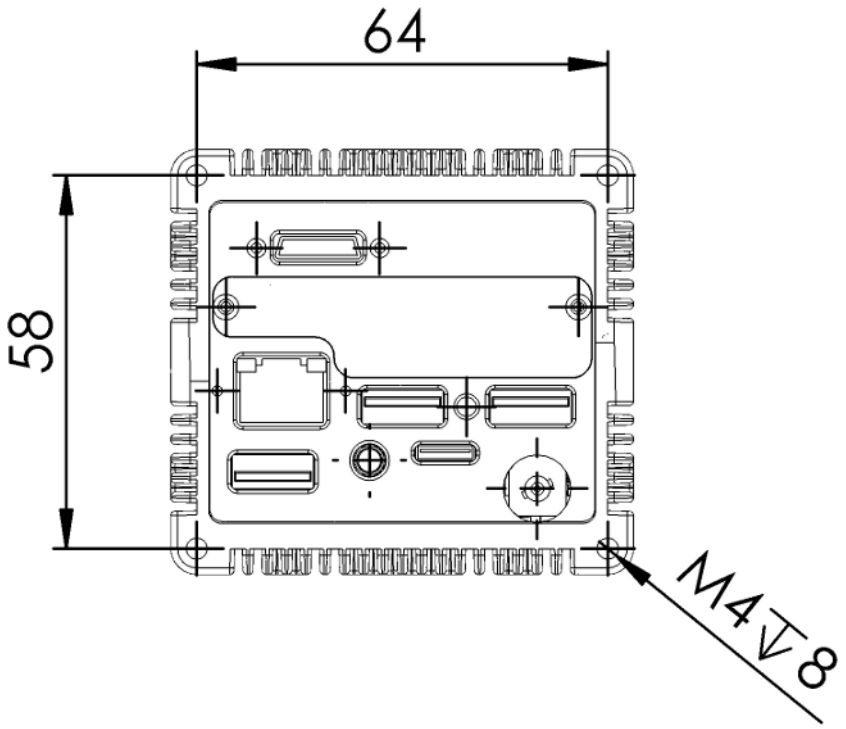


Figure 3.7. CheckSight dimensions, back view

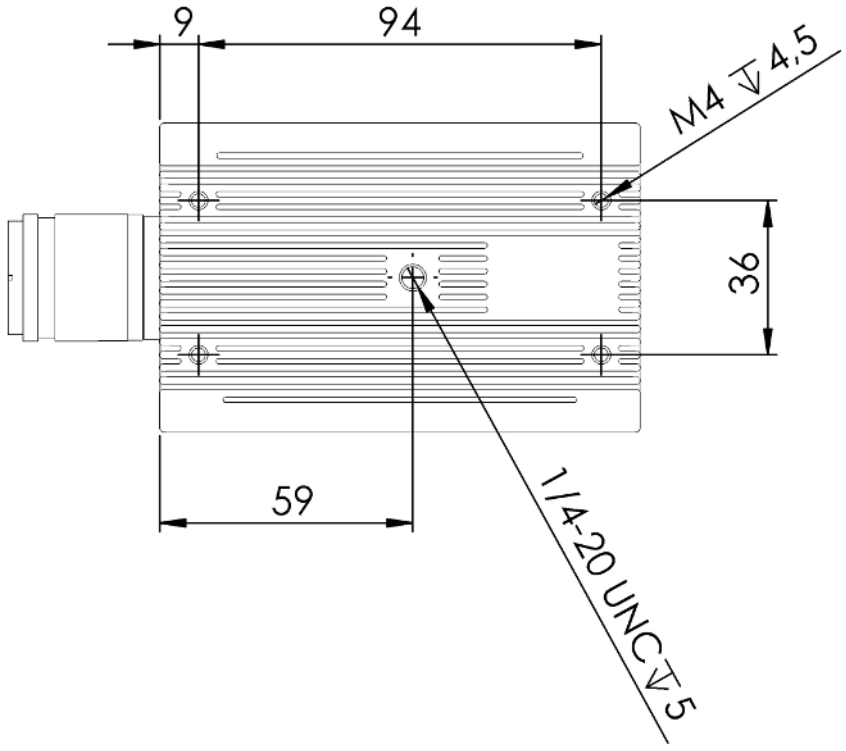


Figure 3.8. CheckSight dimensions, top-down view

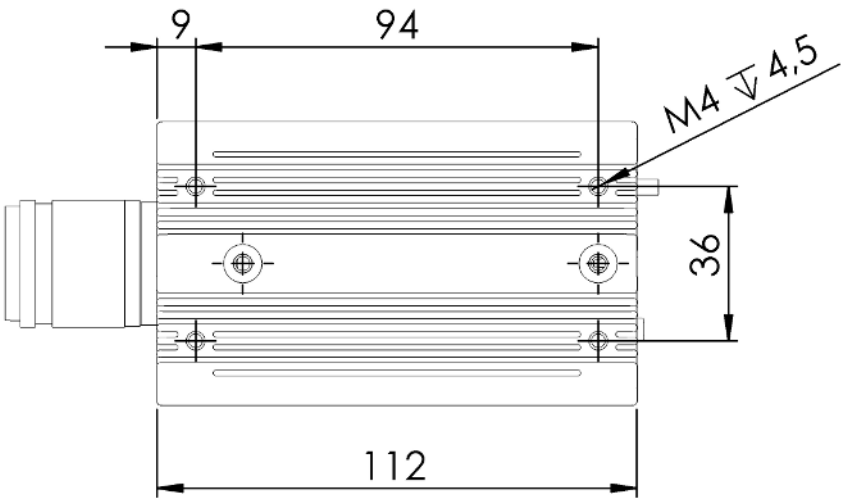


Figure 3.9. CheckSight dimensions, side view

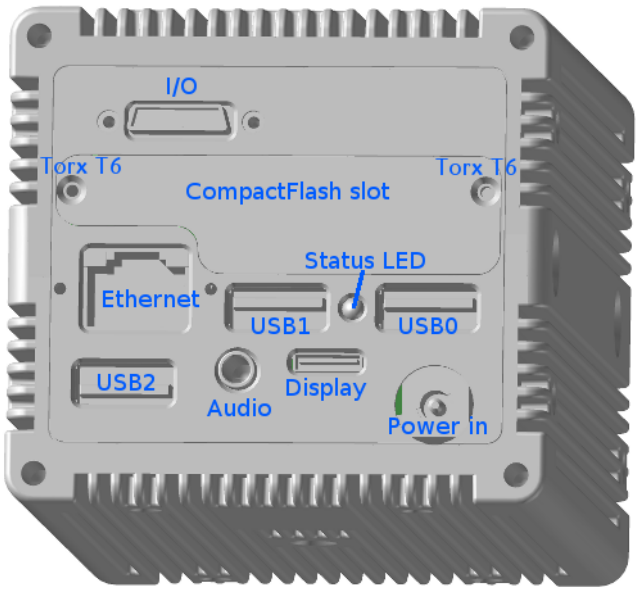


Figure 3.10. CheckSight connector positions

3.4. Mounting points

The mounting points' position and size is illustrated in sections Section 3.3, “Dimensions” [p. 27] and Section 3.1, “Appearance overview” [p. 25].

The following mounting points are available:

- Four M4×8 points in front and back panel of the camera.
- Four M4×4.5 points at each side of the camera.
- One standard tripod mounting point (1/4"-20UNC, 5mm) at the top and bottom sides of the camera.

3.5. Lens mount, optical head

All the models are equipped with a C-mount optical head (1–inch thread diameter, 32 threads per inch, 17.526 mm flange back). CS-mount version (identical, only with 12.5 mm flange back) is available on request for high volume OEM customers.

Few different camera head versions are used for the individual cameras to compensate for different shape (thickness) of individual sensor modules. This guarantees proper flange back 17.526 mm (distance between lens flange and the focal plane, i.e. sensor surface) as specified for C-mount. This has to be taken in account by OEMs (re)assembling the cameras. The different camera head versions are marked using little holes on side to prevent confusion. Contact Leutron Vision for more details.

3.6. Sensor

3.6.1. Sensor types

3.6.1.1. Progressive scanning

All Leutron Vision cameras provide progressive scan video output. The camera outputs full frame data in sequential order (no field interlacing).

3.6.1.2. CMOS vs. CCD

Leutron Vision series offers models with both CMOS and CCD sensors worlds, providing advantages of both technologies.

While CCD chips have reputation of great image quality and high sensitivity, CMOS imagers usually offer higher resolution and dynamic range. But the decision should always be made camera by camera, application by application. The camera series provides wide choice of CCD and CMOS sensors to cover wide range of applications.

3.6.1.3. CMOS: global vs. rolling shutter

Some of the cameras with large CMOS sensors are equipped with a rolling shutter. In contrary to the global shutter, where all camera pixels are exposed at exactly the same time, with rolling shutter the exposure of every single line is slightly time-shifted relative to the previous line. This requires some special considerations, especially when capturing moving objects. Flash illumination is desirable.

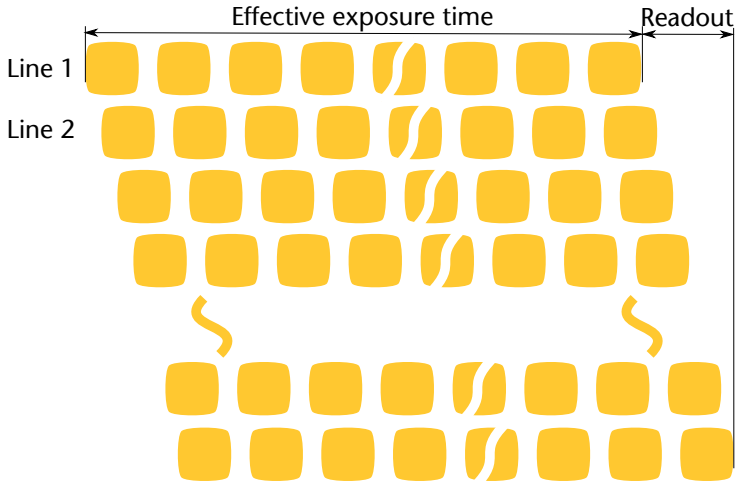


Figure 3.11. Rolling shutter

3.6.1.4. CCD readout types

Most of the CCD sensors are designed with interline transfer readout. These sensors have photo-sensitive pixels organized in columns with light-shielded registers in between them. Upon exposure end the charge from the photosensitive cells is quickly transferred to these storage registers. It is then read out from these registers sequentially, line by line, pixel by pixel and sent to the camera output.

One of the cameras is an exception from this rule, possessing a sensor with frame transfer readout. Such chip is divided into two distinct regions, one photosensitive, the other light-shielded. Upon the exposure end the charge is quickly transferred from the photosensitive area to the storage part of the sensor. Then the image is sent sequentially to the output as usual.

3.7. IR-cut (or other) filter

All the cameras have by default an IR-cut filter mounted on top of the sensor. User has however an option to choose from different configurations:

- IR-cut filter mounted on top of the sensor (default). CCD based cameras come with a 720nm filter, CMOS based ones with 640nm filter.
- Clear glass filter (of the same size as the original IR-cut filter) mounted on top of the sensor. Check, whether you don't need special lens designed for specifically for use without IR-cut filter (common lens might not focus properly in the infrared band or show other problems).
- Daylight blocking filter.
- Without any sensor cover mounted. This has impact on optical properties of the camera — the C-mount flange distance from the sensor is accurately adjusted for the use of the IR-cut filter and removing the filter decreases the length of the optical path. With some lenses and working area setups it can become impossible to focus properly.

User should understand what impact such options can have on the image quality and lens choice and choose option best suiting given application.

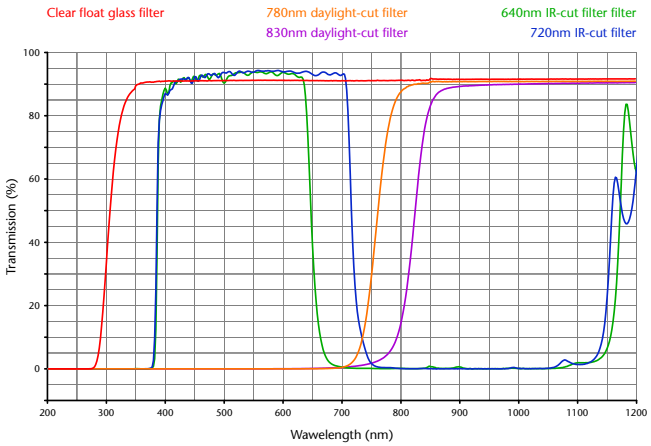


Figure 3.12. Spectral characteristics of individual filter types

3.7.1. Changing or removing the filter

The filter is accessible from outside the camera and user can change it without loose of warranty. It is very important to do this operation with maximum care so that the image sensor cannot be damaged.

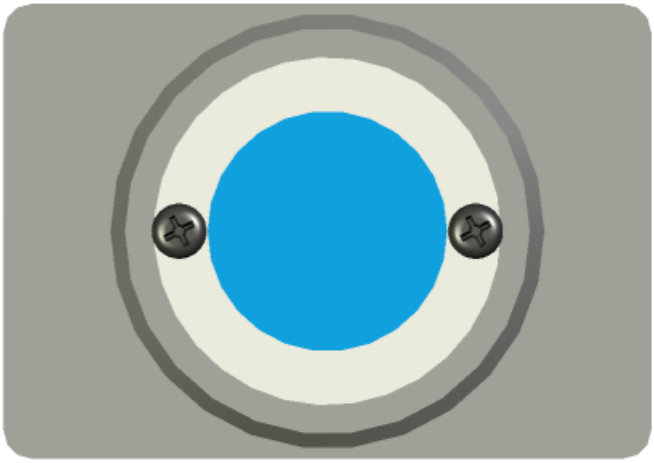


Figure 3.13. IR-cut filter fixing screws

If exchanging the filter for another type, it is highly recommended to choose a filter of the same size, not to change the optical properties of the system. Removing the filter completely will change the length of the optical path which can lead to focusing problems in some scenarios.

To mount/dismount the filter, you will need a TORX screwdriver number 6 (note that some of the first units used hexagonal screws instead). The fixing screws are M2×3.

It's important to keep in mind that when changing or removing the filter, not only can be the sensor damaged, but also dust or other dirt can get inside the camera and especially on top of the sensor itself. It is necessary to perform such operation in appropriately clean environment and using suitable tools preventing any sensor dust pollution.

3.8. Input and output signals

This section describes all types of input and output signals available on CheckSight cameras, including their characteristics and when necessary including connection examples. Location and pinouts of the connectors carrying the signals is documented in Section 3.3, “Dimensions” [p. 27] and Section 3.9, “Connector and cable description” [p. 37].

3.8.1. DC power input

CheckSight cameras are designed for power supply input range of 10—26 V. The power consumption strongly depends on the connected peripherals — devices being powered from CheckSight through the USB or other interfaces. The power consumption of the CheckSight itself is typically 12 W, while the total power consumption, including the connected peripherals, should not exceed 25 W.

A suitable 12V power supply for CheckSight is available from Leutron Vision.

The customers designing an own power supply will find the information about the power input connector in Section 3.9.1, “Power input connector” [p. 37].

3.8.2. PC interfaces

For completness, the PC interfaces of the CheckSight camera are again listed below. The components driving individual principal interfaces are described in Section 3.2, “System architecture” [p. 26].

Ethernet	CheckSight provides a Gigabit Ethernet connectivity (compatible also with 10/100Mb networks).
Display	CheckSight connects with DVI-D compatible displays. The DVI-D interface is available through a custom connector (Section 3.9.6, “Display connector” [p. 40]), adapter cable is available.
USB	3 USB ports are available as standard Type A connectors. The USB power lines are protected and are limited to 500mA per USB device. USB data lines fulfil IEC61000-4-2 ESD protection. Leutron Vision provides USB Type A adapters to avoid space problems if USB devices with big molds are to be plugged into the CheckSight
CompactFlash	Interface for CompactFlash memory cards type I. The CompactFlash slot is connected to primary IDE channel (parallel ATA interface).
Audio	Line output and microphone input is available (Section 3.9.7, “Audio connector” [p. 41]).

3.8.3. Trigger input

Any of the available optocoupler inputs or TTL inputs can be used for triggering. The input line mapping on the camera connectors is described in Section 3.9, “Connector and cable description” [p. 37].

The characteristics and default behavior of these interfaces are described below in Section 3.8.4, “Optocoupler interface” [p. 33] and Section 3.8.5, “TTL I/O's” [p. 36]. Those sections also provide examples how to connect the inputs.

3.8.4. Optocoupler interface

Optically isolated inputs and outputs.

3.8.4.1. Optocoupler basics

An optocoupler is a device using optical path to transfer an electronic signal between two circuits. It basically consists of a photodiode converting the input signal to light and a phototransistor converting the light again to electronic signal.

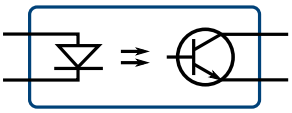


Figure 3.14. Diagram of basic optocoupler function

The optocoupler is useful in situations where one part of the circuit needs to be galvanically isolated from the other part to prevent damage or unwanted interference. It is used to prevent ground loops or to block voltage spikes.

For effective use of the optocoupler it is important to control it using an independent power supply. When the external circuit is galvanically connected to the camera (eg. when reusing the camera's power to control the optocoupler), the optocoupler's protective function is eliminated.

3.8.4.2. Optocoupler inputs

The CheckSight camera is equipped with four optically isolated input lines. One input optocoupler is fully independent, the other three are connected with a common cathode (Section 3.9.2, “I/O connector” [p. 37]) due to limited number of pins on the I/O connector.

The optocoupler input's logical state is understood as logical `0` if no current is flowing through the optocoupler diode, logical `1` if current is flowing through the diode.

The optocoupler is guaranteed to be switched on (logical `1` status) when the voltage applied to the input is within range of 5—24 V (tolerance 10%). It is guaranteed to be switched off (logical `0`) when the applied voltage is under 1.5 V. The input status is undefined between 1.5 and 4.5 V, the input voltage must not exceed the 24 V limit.

The optocoupler inputs have no protection against wrong polarity, be sure to connect them always with correct polarity according to the documented wiring diagrams and connector pinouts.

The nominal input current is 7.5 mA — the input signal must be able to deliver at least this amount of current.

The optocoupler inputs are TTL compatible provided that the signal source is capable of supplying the input current of 7.5 mA.

The optocoupler input lines are equipped with a debouncer to prevent generating unwanted input signals. Its functionality and control options are documented in Section 6.7.9, “Input lines debouncer” [p. 123].

The input lines are using fast optocouplers: the propagation time is under 125 ns, for both rising and falling signal edge (assuming input signal with strongly steep edge). The exact switching speed may vary with temperature and the used input voltage level — the listed specifications are the worst case values.

Parameter	**Value**
Operating voltage	5—24 V
Input current	7.5 mA
External resistor requirement	No
ON voltage level	> 4.5 V
OFF voltage level	< 1.5 V
OFF to ON delay	< 125 ns
ON to OFF delay	< 125 ns
OFF to ON jitter	± 5 ns
ON to OFF jitter	± 5 ns

Table 3.1. Optocoupler input parameters summary

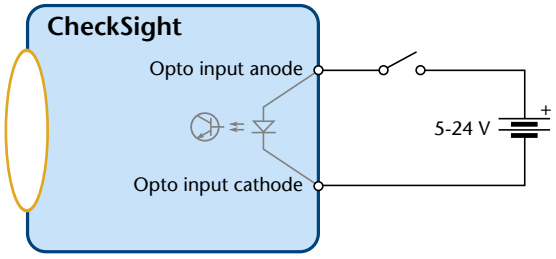


Figure 3.15. Example of using the optocoupler inputs

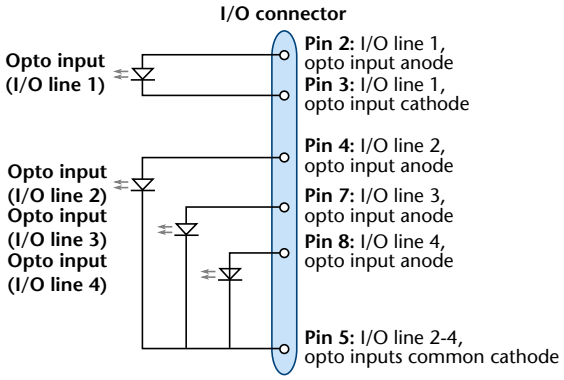


Figure 3.16. Diagram of the internal optocoupler inputs connections

3.8.4.3. Optocoupler outputs

The CheckSight camera is equipped with four optically isolated output lines. One output optocoupler is fully independent, the other three are connected with a common collector (Section 3.9.2, "I/O connector" [p. 37]) due to limited number of pins on the I/O connector.

After power up, the optocoupler outputs are in high impedance state and they remain in that state until reconfigured from the software interface. Whenever the optocoupler output is disconnected in the SW interface, it gets in high impedance status.

The optocoupler output's logical state is understood as logical `0` when its output transistor is in high impedance, logical `1` if it is in low impedance mode.

The optocoupler output work with operating voltage range of 5—24 V.

The maximal switched output current is 100 mA per optocoupler, higher current could damage the optocoupler output circuit.

The rising edge (switching to logical `1`) propagation time is 750 ns, the falling edge propagation time is under 25 µs. The exact switching speed may vary with temperature and the used input voltage level — the listed specifications are the worst case values.

Parameter	Value
Operating voltage	5—24 V
Output current	< 100 mA
External resistor requirement	Yes, output current must be limited to 100 mA
OFF to ON delay	750 ns
OFF to ON rise time (10 to 90 %)	1.0—4.0 µs
ON to OFF delay	2.5—25 µs
ON to OFF fall time (10 to 90 %)	9.0—140 µs
OFF to ON jitter	± 20 ns
ON to OFF jitter	± 0.1 µs

Table 3.2. Optocoupler output parameters summary

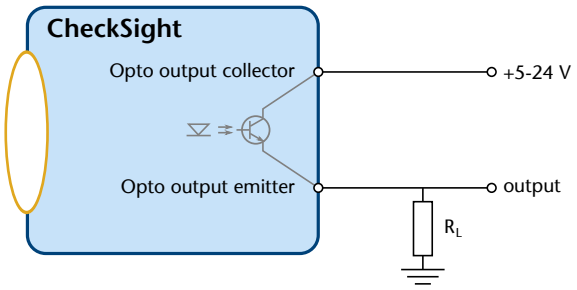


Figure 3.17. Example of using the optocoupler outputs

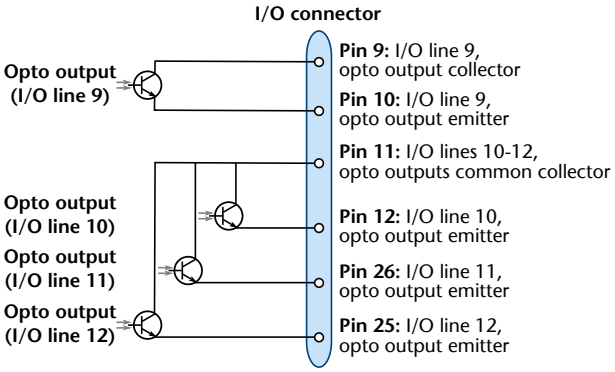

I/O connector
Opto output (I/O line 9)
Opto output (I/O line 10)
Opto output (I/O line 11)
Opto output (I/O line 12)
Pin 9: I/O line 9, opto output collector
Pin 10: I/O line 9, opto output emitter
Pin 11: I/O lines 10-12, opto outputs common collector
Pin 12: I/O line 10, opto output emitter
Pin 26: I/O line 11, opto output emitter
Pin 25: I/O line 12, opto output emitter

Figure 3.18. Diagram of the internal optocoupler outputs connections

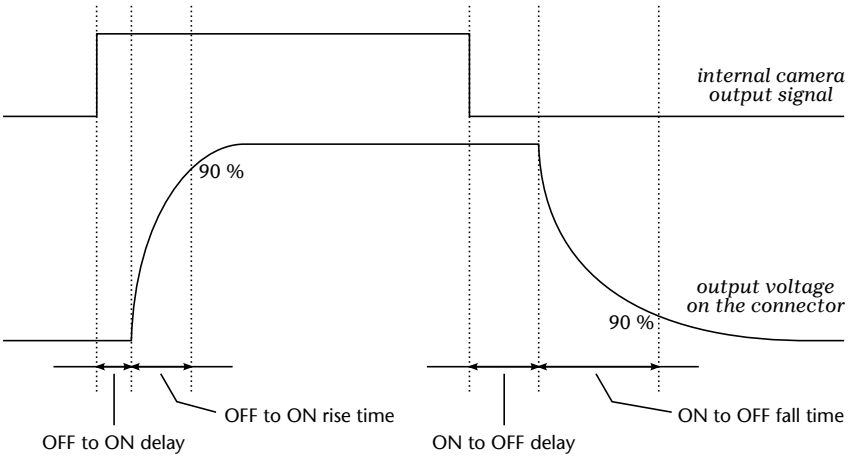


Figure 3.19. CheckSight output optocoupler timing diagram

3.8.5. TTL I/O's

Standard 5V TTL (transistor-transistor logic) interface.

After power up, the TTL outputs are driven high — and they remain driven high until reconfigured from the software interface. Whenever the TTL output is disconnected in the SW interface, it gets in driven high status. TTL inputs are pulled up to 5V inside the camera to guarantee a stable input in case nothing is connected. To generate a steep edge the TTL input driver should be capable of driving and sinking 10mA.

The TTL input's and output's logical state is understood as logical `1` when it's driven high (5V) and logical `0` when it's not driven (0V).

The TTL input lines are equipped with a debouncer to prevent generating unwanted input signals. Its functionality and control options are documented in Section 6.7.9, “Input lines debouncer” [p. 123].

The delays listed in the table are setup dependent mean values, while the jitters are maximal deviations (±) from those mean values.

Parameter	Value
Operating voltage	5V±10% (absolute maximum 6.5V)
ON voltage level	> 3.5 V
OFF voltage level	< 1.5 V
High level output current	-32mA max / high level at 3.8V min
Low level output current	32mA max / low level at 0.55V max
Input to sensor trigger delay	19 µs (jitter +13/-2 µs)
TTL-in to TTL-out delay	32 µs (jitter +10/-2 µs)

Table 3.3. TTL input/output parameters summary

3.8.6. Serial port interface

Standard RS-232 communication interface, accessible directly from the camera's feature tree.

Configurable communication parameters are: baud rate, parity, data bits, stop bits.

3.9. Connector and cable description

All the grounds from individual connectors are connected inside the camera.

3.9.1. Power input connector

Standard 2pin power jack connector interfaces to standard DC power supply over 5.5×2.5mm plug. Locking prevents accidental plug-to-jack decoupling.

The central pin (tip) of the connector carries the + power input, the outer pin (sleeve) connects the ground.

Miniature locking power plugs may be ordered from Leutron Vision

Use always power supply fulfilling all the specifications listed in Section 3.8.1, “DC power input” [p. 32].

3.9.2. I/O connector

A 26pin SDR female connector for digital I/O (optocoupler and TTL inputs/outputs) and serial port (RS-232) interface.

An adapter cable with D-sub 25-pin female connector is available. The table lists also pin numbers on the D-sub connector. The cable construction is described below.

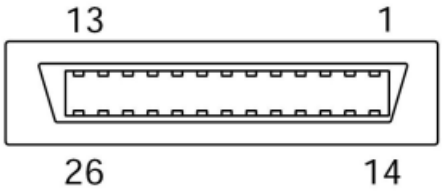


Figure 3.20. I/O connector (female), pinout

Pin	Signal	Level	D-sub 25
1	Ground	Ground	12
2	I/O line 1: optocoupler input anode	Ext power +5—+24V	1
3	I/O line 1: optocoupler input cathode	Ext. ground	2
4	I/O line 2: optocoupler input anode	Ext power +5—+24V	3
5	I/O lines 2—4: optocoupler inputs common cathode	Ext. ground	4
6	Ground	Ground	5
7	I/O line 3: optocoupler input anode	Ext power +5—+24V	6
8	I/O line 4: optocoupler input anode	Ext power +5—+24V	7
9	I/O line 9: optocoupler output collector	Ext power +5—+24V	8
10	I/O line 9: optocoupler output emitter	Ext. output	9
11	I/O line 10—12: optocoupler outputs common collector	Ext power +5—+24V	10
12	I/O line 10: optocoupler output emitter	Ext. output	11
13	Ground	Ground	n.c.
14	Reserved — DO NOT CONNECT!	n/a	13
15	Reserved — DO NOT CONNECT!	n/a	14
16	RS-232 RxD (serial data input)	RS-232	15
17	RS-232 TxD (serial data output)	RS-232	16
18	I/O line 26: TTL output	5V TTL	17
19	I/O line 18: TTL input	5V TTL	18
20	Reserved — DO NOT CONNECT!	n/a	19
21	I/O line 25: TTL output	5V TTL	20
22	I/O line 17: TTL input	5V TTL	21
23	Ground	Ground	22
24	Reserved — DO NOT CONNECT!	n/a	23
25	I/O line 12: optocoupler output emitter	Ext. output	24
26	I/O line 11: optocoupler output emitter	Ext. output	25

Table 3.4. I/O connector, pinout (including pin numbers on the SDR-26 to D-sub-25 adapter)

3.9.2.1. SDR-26 connector adapter cable

An adapter cable to interface the camera's 26-pin SDR connector is available. The other side of the cable holds a D-sub 25-pin female connector. The D-sub connector can either be used directly

or the cable can be opened and used to interface another custom connector type. The construction of the cable is described in the following figure.

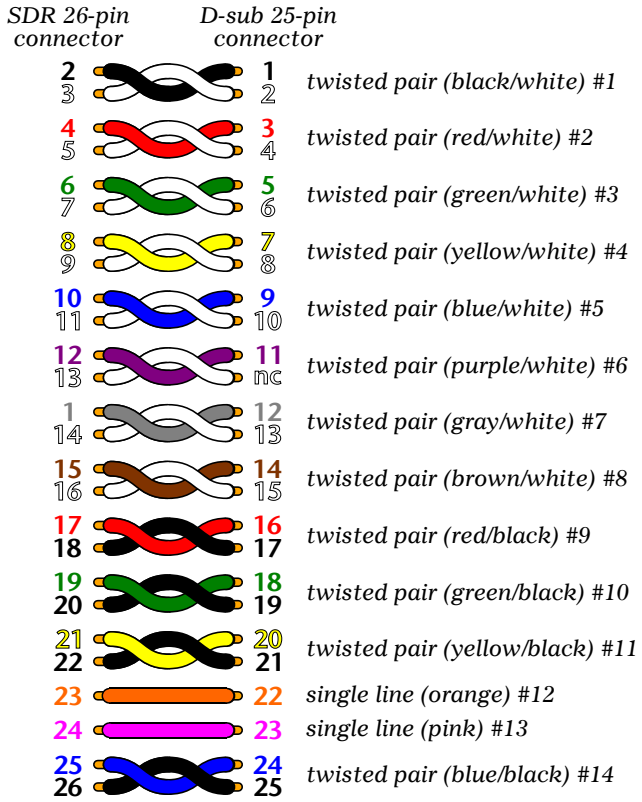

SDR 26-pin connector
D-sub 25-pin connector
2 3 — 1 2 twisted pair (black/white) #1
4 5 — 3 4 twisted pair (red/white) #2
6 7 — 5 6 twisted pair (green/white) #3
8 9 — 7 8 twisted pair (yellow/white) #4
10 11 — 9 10 twisted pair (blue/white) #5
12 13 — 11 nc twisted pair (purple/white) #6
1 14 — 12 13 twisted pair (gray/white) #7
15 16 — 14 15 twisted pair (brown/white) #8
17 18 — 16 17 twisted pair (red/black) #9
19 20 — 18 19 twisted pair (green/black) #10
21 22 — 20 21 twisted pair (yellow/black) #11
23 — 22 single line (orange) #12
24 — 23 single line (pink) #13
25 26 — 24 25 twisted pair (blue/black) #14

Figure 3.21. 26-pin SDR to 25-pin D-sub adapter cable

3.9.3. Ethernet connector

A standard Ethernet connector, 8P8C modular plug, often also referred to as RJ-45. Refer to the corresponding Ethernet specifications for more details.

The connector is equipped with two standard status LED's.

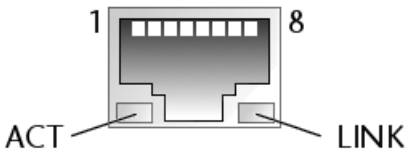


Figure 3.22. Ethernet connector

Pin	Wire color code	Signal (1000BASE-T)	Signal (10BASE-T, 100BASE_TX)
1	white/green	BI_DA+	TX+
2	green	BI_DA-	TX-
3	white/orange	BI_DB+	RX+
4	blue	BI_DC+	NC
5	white/blue	BI_DC-	NC
6	orange	BI_DB-	RX-
7	white/brown	BI_DD+	NC

Pin	Wire color code	Signal (1000BASE-T)	Signal (10BASE-T, 100BASE_TX)
8	brown	BI_DD-	NC

Table 3.5. Ethernet connector, pinout

The color scheme and pinout shown in the table above applies for the straight-through cables (cable wiring standard EIA-568-A). For cross over cable, wire pairs 1–3, 2–6, 4–7 and 5–8 had to be crossed. Refer to official Ethernet connector and cabling specifications in case of any doubts.

3.9.4. USB connectors

3 standard USB connectors (USB Type A) are accessible directly from the camera's back panel, see Section 3.3, “Dimensions” [p. 27]. The connectors are marked as USB0, USB1 and USB2.

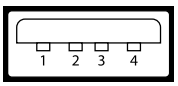


Figure 3.23. Standard USB connector, type A

Pin	Signal
1	USB power supply (5V, max 500mA)
2	USB D- (data transfer)
3	USB D+ (data transfer)
4	Ground

Table 3.6. USB 0—2 connectors (type A), pinout

3.9.5. Data storage device connectors

A standard CompactFlash type I connector is available for industrial grade CompactFlash cards.

The CompactFlash slot is covered by a cover plate fixed by two screws (see Section 3.3, “Dimensions” [p. 27] and Section 3.1, “Appearance overview” [p. 25]). Use Torx screwdriver size T6 to loosen these screws and get access to the CompactFlash slot.

3.9.6. Display connector

A custom miniature connector, carrying the DVI-D (digital only) display signals. Adapter cable with standard DVI-D connector is available from Leutron Vision. Note that use of the custom connector was necessary due to space constraints.

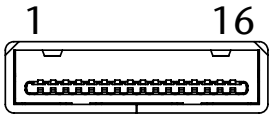


Figure 3.24. Display connector

Pin	Signal
1	DVI DDC data
2	DVI DDC clock

Pin	Signal
3	Ground
4	DVI TMDS clock -
5	DVI TMDS clock +
6	Ground
7	DVI TMDS Data 0- (digital blue)
8	DVI TMDS Data 0+ (digital blue)
9	DVI TMDS Data 1- (digital green)
10	DVI TMDS Data 1+ (digital green)
11	Ground
12	DVI TMDS Data 2- (digital red)
13	DVI TMDS Data 2+ (digital red)
14	Ground
15	+5V
16	DVI Hotplug detect

Table 3.7. Display connector, pinout

3.9.7. Audio connector

A standard 3-pin 3.5 mm audio jack (female), also known as “TRS” (tip-ring-sleeve) connector.

An adapter cable splitting the connector to line out (green) and microphone in (pink), each realized as a separate standard 3.5 mm female audio jack is available from Leutron Vision.

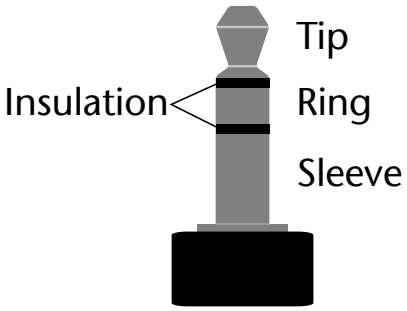


Figure 3.25. Audio connector (TRS jack), documented on the male connector version

Pin	Signal (CheckSight backplane)	Signal (adapter, line out)	Signal (adaptder, mic in)
1 (“Sleeve”)	Ground	Ground	Ground
2 (“Ring”)	Microphone in	Line out (mono)	n/a
3 (“Tip”)	Line out (mono)	n/a	Microphone in

Table 3.8. Audio connector, pinout

3.10. Status LED

The CheckSight are equipped with a LED providing basic visual feedback about the current status of the device.

When the device is powered up, the LED produces a stable green light.

*The Ethernet connector contains additional two LED's providing status information about the Ethernet connection, refer to**Section 3.9.3, “Ethernet connector” [p. 39]**.*

4. Software installation

4.1. Operating system

CheckSight supports following operating systems:

- Windows: Windows XP and later.
- Windows Embedded: Windows Embedded Standard and newer editions of the Windows Embedded line.
- Linux: contact Leutron Vision to get the list of officially supported distributions and other system requirements. In general, any reasonably new distribution based on kernel 2.6 should work fine.
- Contact Leutron Vision for information of possible support of other operating systems.

Follow the installation instructions from the documentation of your operating system.

Use a USB CD-drive to install the operating system.

After basic operating system installation, you might need to install additional specific device drivers for the CheckSight system or perform additional configuration to get optimal performance. Corresponding instructions are provided below.

4.2. Windows

The description and screenshots are provided for Windows XP, but are generally applicable to any Windows version.

All the necessary device drivers are available from our download area[1]. We recommend to install them in the order how they are presented in the sections below.

4.2.1. Chipset drivers

Navigate to the folder with chipset driver files, execute `setup.exe` and follow the instructions. After installation, reboot the system.

Intel® Chipset Device Software
Intel® Chipset Device Software
intel
Welcome to the Setup Program
This setup program will install the Intel® Chipset Device Software onto this computer. It is strongly recommended that you exit all programs before continuing.
< Back
Next >
Cancel
Intel® Installation Framework

Figure 4.1. Chipset drivers installation

[1] http://www.leutron.com/support-downloads/download-area/

Intel® Chipset Device Software
Intel® Chipset Device Software
Setup Progress
Please wait while the following setup operations are performed:
Installing Driver: Intel(R) SCH Family USB Universal Host Controller - 8114
Version: 8.8.0.1001
Installing Driver: Intel(R) SCH Family USB Universal Host Controller - 8115
Version: 8.8.0.1001
Installing Driver: Intel(R) SCH Family USB Universal Host Controller - 8116
Version: 8.8.0.1001
Installing Driver: Intel(R) SCH Family USB2 Enhanced Host Controller - 8117
Version: 8.8.0.1001
Click Next to continue.
Next
Intel® Installation Framework

Figure 4.2. Chipset drivers' installation in progress

4.2.2. Display driver

Navigate to the folder with display driver files, execute `setup.exe` and follow the instructions. After installation, reboot the system.

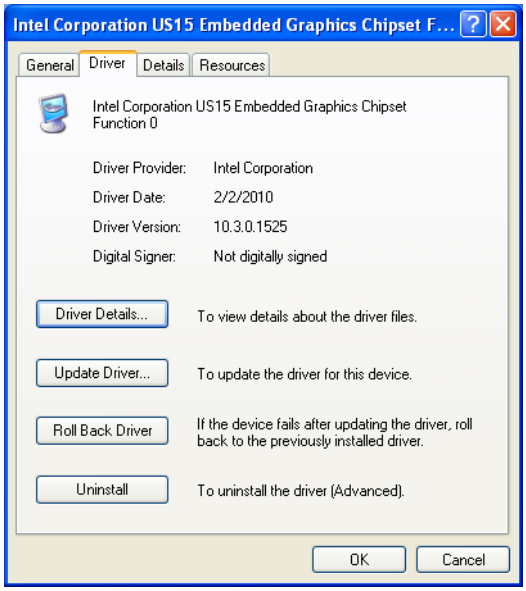


Figure 4.3. Display driver properties

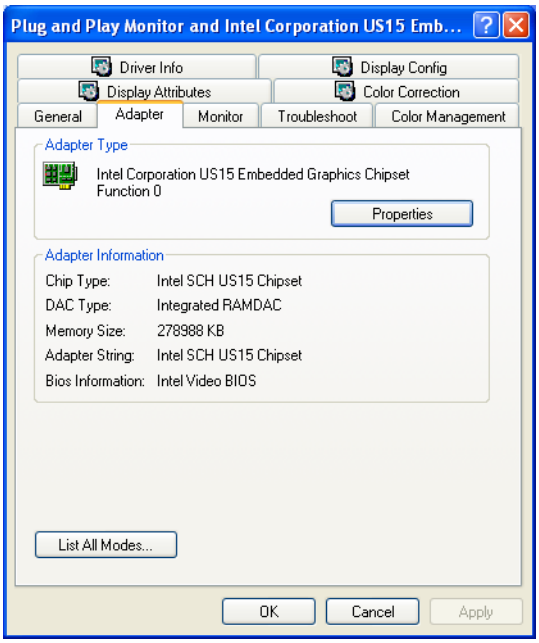


Figure 4.4. Display properties, configuration options

4.2.3. Network driver

Navigate to the folder with network driver files, execute `setup.exe` and follow the instructions. After installation, reboot the system.

To optimize network performance, you might want to edit the driver properties (accessible from the Device manager). Make sure the network configuration is compatible with other devices in your network.

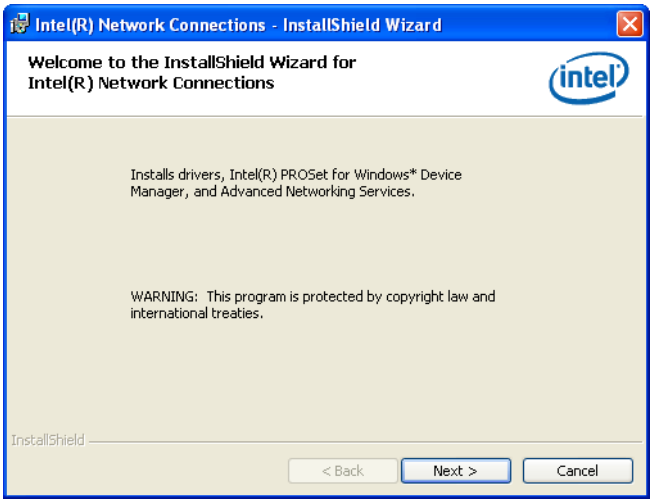

Intel(R) Network Connections - InstallShield Wizard
Welcome to the InstallShield Wizard for Intel(R) Network Connections
intel
Installs drivers, Intel(R) PROSet for Windows* Device Manager, and Advanced Networking Services.
WARNING: This program is protected by copyright law and international treaties.
InstallShield
< Back
Next >
Cancel

Figure 4.5. Network driver installation

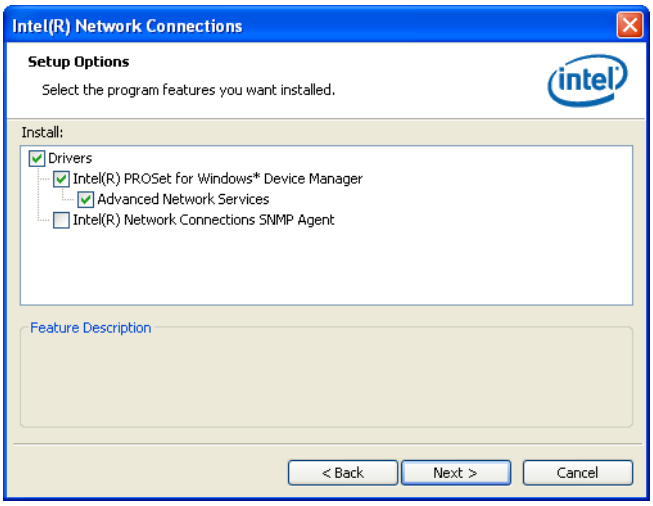


Figure 4.6. Network driver installation options

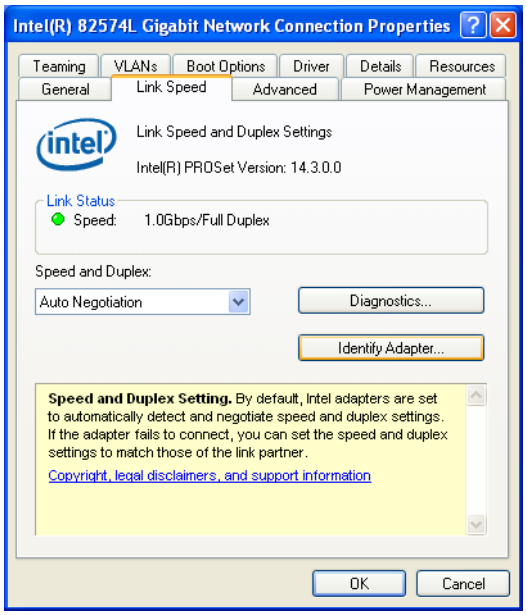


Figure 4.7. Network driver, speed and duplex settings

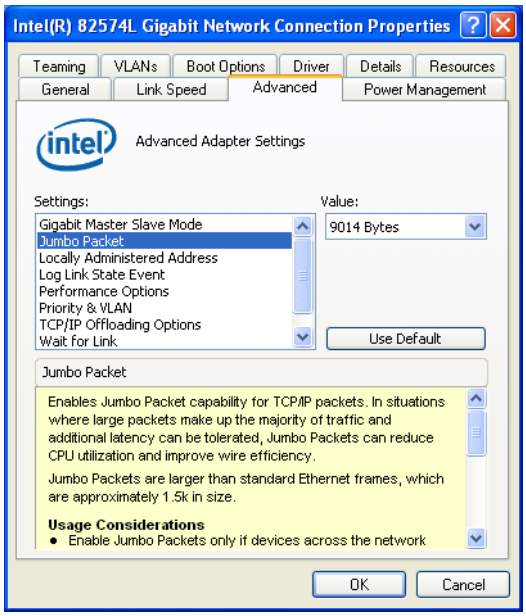


Figure 4.8. Network driver, jumbo packet configuration

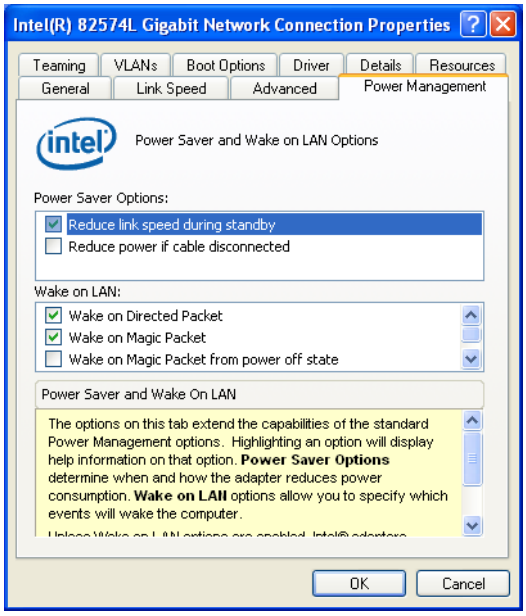

Intel(R) 82574L Gigabit Network Connection Properties
Teaming
VLANs
Boot Options
Driver
Details
Resources
General
Link Speed
Advanced
Power Management
Power Saver and Wake on LAN Options
Power Saver Options:
Reduce link speed during standby
Reduce power if cable disconnected
Wake on LAN:
Wake on Directed Packet
Wake on Magic Packet
Wake on Magic Packet from power off state
Power Saver and Wake On LAN
The options on this tab extend the capabilities of the standard Power Management options. Highlighting an option will display help information on that option. Power Saver Options determine when and how the adapter reduces power consumption. Wake on LAN options allow you to specify which events will wake the computer.
OK
Cancel

Figure 4.9. Network driver, power saving options

4.2.4. Audio driver

Navigate to the folder with audio driver files, execute `setup.exe` and follow the instructions. After installation, reboot the system.

The driver comes with additional control utilities which allow to adjust the audio configuration.

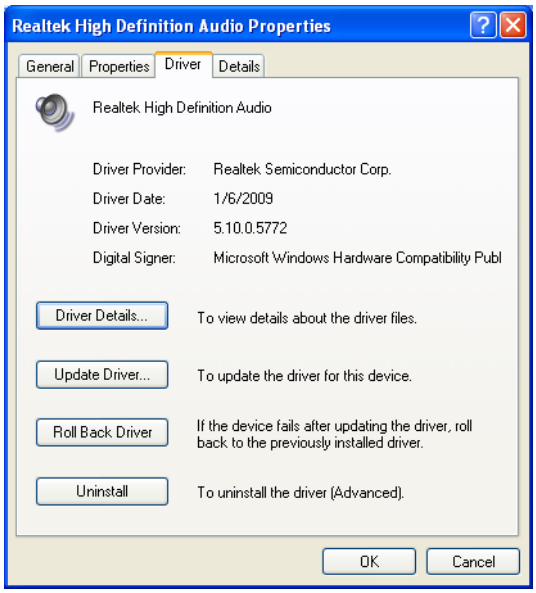


Figure 4.10. Audio driver properties

4.2.5. Power management settings

Depending on how the CheckSight system will be used, it might be necessary to adjust the “Power options”, configurable through the Control panel. To prevent the system going standby when working in standalone mode (no user activity), use the “Always on” power scheme and disable hibernation.

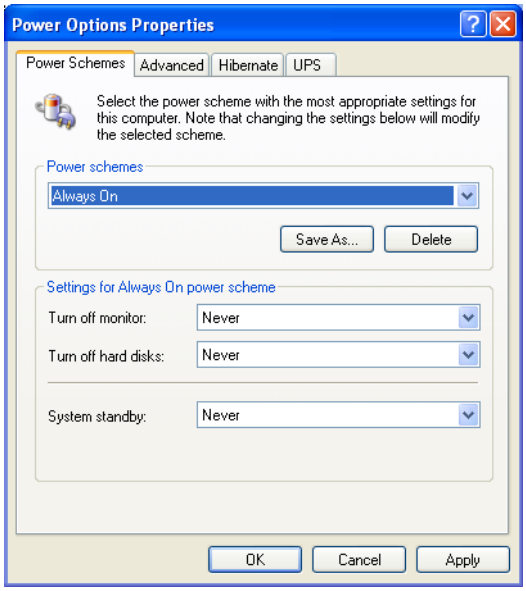


Figure 4.11. Power options, power scheme “always on”

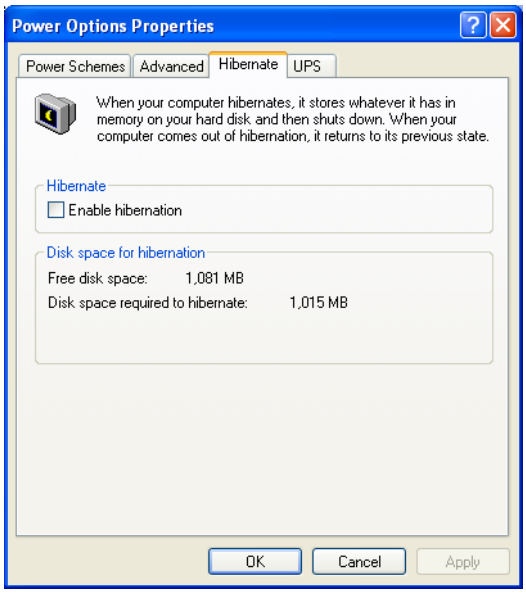

Power Options Properties
Power Schemes
Advanced
Hibernate
UPS
When your computer hibernates, it stores whatever it has in memory on your hard disk and then shuts down. When your computer comes out of hibernation, it returns to its previous state.
Hibernate
Enable hibernation
Disk space for hibernation
Free disk space: 1,081 MB
Disk space required to hibernate: 1,015 MB
OK
Cancel
Apply

Figure 4.12. Power options, hibernation disabled

4.3. Windows Embedded Standard

Embedded family of operating system is suitable for small platforms with various integrated modern technologies. It's componentized design and advanced features allows hardware resources to be utilized optimally.

4.3.1. Requirements

1. Please check a minimal requirements for selected version of operating system listed below.

   Note: This requirement become optional in case your CheckSight has an on-board compact flash (at least 4GB).

2. Monitor, USB mouse, USB keyboard, USB-DVD drive

4.3.2. Installation procedure for systems distributed as ISO files

1. Burn ISO file on DVD.
2. Prepare a CheckSight for installation. Connect all required devices, insert DVD from step 1 into USB-DVD drive.
3. When CheckSight logo appears, press F11 key to get into boot menu.
4. Select a USB-DVD drive as a boot device and press ENTER.
5. Let Windows installation environment to boot up and wait for a command prompt.
6. Use command prompt to start installation.

   a. In case a compact flash is factory clean (it hasn't been partitioned yet) install the system by:

   ```
   install.cmd
   ```

   b. If a compact flash contains old data start installation by:

   ```
   install.cmd /f
   ```

Please note that all data on the first disk will be overwritten!

4.3.3. List of distributed systems

1. Windows Embedded Standard v1.5.101206.iso

Name	Windows XP Embedded (Standard 2009/English/CheckSight)
Version	1.5.101206
Article number	998000
Related files	`CS_WinXPe-v1.5.101206.iso`
Description	Version of Windows XP Embedded aimed to support developers who need to start to work on application immediately without loosing a time by preparing suitable build environment or for system recovery.
Minimal requirements	System needs at least 4GB of space on compact flash card for self installation
User accounts	username: checksight, password: 1234 username: Administrator, password: 1234
Default partition scheme	One partition 3.4GB (rest of compact flash remains un-partitioned), NTFS, FBWF filters installed but not enabled. 1.76GB occupied by operating system after installation, 1.64GB is free for use by user.
Software installed	• Simplon v1.00.030 • Microsoft .NET Framework v3.5 SP1
Drivers installed	• US15 Intel Embedded Media and Graphics Driver(v6.14.01.1742 • Realtek High Definition Audio v5.10.0.5953 • Intel(R) 82574L Gigabit Network Connection v10.6.15.0

Table 4.1. WinXPe v1.5.101206

4.3.4. Useful hints

Network high performance settings. In case you need to work with Gigabit cameras it is suitable to set up network adapter to be able to handle heavy traffic without overloading of CPU. Following parameters have significant influence on network performance

- Jumbo frame length
- Inter packet delay
- Filter driver

For more information please refer to PicSight-GigE manual

Windows firewall. A firewall settings is a typical problem in case that user has successfully connected the camera but he cannot see any images after star of acquisition.

The fastest way how to verify if firewall is a guilty one is to switch it off and start acquisition again. If you can see images you can enable the firewall and start to play with exceptions allowing UDP packets to traverse through the firewall.

For more information about this topic please refer to PicSight-GigE manual.

FBWF - File Based Write Filters. FBWF is a tool allowing to save a compact flash lifetime by redirecting unwanted writes into operating memory rather than directly into compact flash.

Filters are controlled by command line utility called `fbwfmgr.exe`. In default installation the FBFW filters for drive `C:` are installed but not enabled.

- You can enable FBWF by following sequence of commands:

  ```
  fbwfmgr /enable
  fbwfmgr /addvolume c:
  ```

  After next reboot the drive C: will be protected by FBFW.

- You can add an exclusion for certain directory. In this case all changes applied to the excluded directory will by written directly in storage media.

  E.g. adding write exception for c:\Temp directory:

  ```
  fbwfmgr /addexclusion c: \Temp
  ```

- To remove FBWF protection from drive `c:` run:

  ```
  fbwfmgr /removevolume c: 1
  ```

  1 ... all previously set exclusions will be forgotten

  0 ... all previously set exclusions will be stored and automatically applied by next enabling of FBWF

  ```
  fbwfmgr /disable
  ```

  or

  `fbwfmgr /help /addexclusion` (if you need to get more information about this topic)

4.4. Linux

Because of an existence of many Linux distributions it is not possible to cover all installation and configuration details for each one. In the following section you can find useful information how to modify hardware and software settings to gain a full profit from your Linux installation.

In general every distribution based on kernel 2.6 should work fine with CheckSight — it means the main hardware on Chipset (USB Controller, IDE Controller, VGA, Network Adapter) which is essential for run of the operating system is properly recognized during setup. Support of further features which can improve graphics acceleration and CheckSight scalability (e.g. SDIO) are distribution dependent.

Following list refers to these distributions which were successfully tested with CheckSight and Simplon:

- Ubuntu 10.04 LTS (graphics driver is not available for this version)
- Ubuntu 10.10

- ...

Note: List in not enclosed and further distribution are going to be added.

4.4.1. Chipset, network and audio drivers

Hardware is supported in kernel 2.6 and higher.

4.4.2. Display driver

CheckSight offers graphics acceleration capability to operating system. Drivers Intel GMA 500 chip are under development process but official version for Linux hasn't been released yet. Nevertheless Linux is able to use graphics unit in standard VGA mode. In this case a maximal graphics resolution the user can use depends on BIOS setting.

4.4.3. Recommendations

- **VGA memory.** For maximal graphics performance a presence of Intel GMA 500 driver in operation system is needed. If the driver is missing then kernel will initialize only VGA Controller. In this case an amount of VGA m.emory reserved for display has a significant influence on maximal graphics resolutions which can be set by operating system. If you have a problem to set more than 800 x 600 pixels in 32 color depth in VGA mode than enter into BIOS menu (entering menu by pressing F11 key when CheckSight logo appears after restart) and reserve more memory for VGA (maximum value is 8 MB). This will allow you to set 1024 x 768 pixels in 32 bit color depth.

  BIOS Menu → Chipset → North Bridge → Integrated Graphics Mode Select → [Enabled, 8M]

- **SDIO controller.** Some Linux distributions have a problem to handle SDIO controller. As a side effect they can produce a big amount of information into log files. Increasing of log information may occupy valuable place on Compact Flash card. One way is to disable SDIO Controller in case the user doesn't need it. Note:BIOS version 105 and higher provide a hot fix for this issue.

  BIOS Menu → Chipset → South Bridge → SDIO Controller → [Disabled]

- **GRUB configuration.** Linux kernel must know that CheckSight Chipset is missing Super — IO chip otherwise a `reboot` command will not work and CheckSight stop responding right before restart is performed. User can inform kernel by passing `reboot=b` parameter at the time when operating system is booting up. Following procedure is specific for Grub 2.x installed on Ubuntu 10.04 LTS. Please note that configuration may differ from distribution to distribution.

  1. Open `/etc/default/grub`and add `reboot=b` parameter into `GRUB_CMD_LINE_LINUX_DEFAULT` key.

     Modified line should look like: *`GRUB_CMD_LINE_LINUX_DEFAULT="reboot=b ..."`*, where *"..."* may represents further commands (spaced by white character).

  2. Call `update-grub` command to make changes in Grub permanent. Operating system will be able to restart CheckSight after reboot.

- **LOG files.** It is useful to safe a lifetime of Compact Flash by reducing amount of write accesses to it as much as possible. It can be done by using of special file system designed for Compact Flash devices like JFFS, YAFFS or UBIFS. Other and much simpler way is to disable unnecessary logging. Please refer to documentation of your distribution for more information.

4.5. Simplon

Following sections provide installation instructions for Simplon on the Windows and Linux operating systems. More details regarding the installation options (other operating systems, custom installation, etc.), Simplon configuration and other related topics can be found in Simplon getting started.

4.5.1. Installation in Windows

4.5.1.1. System requirements

The Simplon for Windows does not have any particular system requirements, the installer will install all the necessary dependencies, including the .NET Framework runtime. Simplon is compatible with all "recent" Windows versions, including Windows 7, Windows Vista or Windows XP. Note that Windows versions older than Windows 2000 are not supported.

The Simplon package installation should be performed by a user with administrator rights.

4.5.1.2. Installation media

The Simplon package can be installed from two types of installation media:

- If you obtained the installation CD-ROM, insert it to the CD-ROM drive. The installation will start automatically.
- If you don't have the installation CD-ROM or if you wish to install a newer Simplon version, you can download the installer from the download area[2] on our website. The installer file name follows pattern `Simplon`*`xyyzzz`*`.exe`, where *`xyyzzz`* stands for Simplon version.

  Execute the installer. Depending on security adjustments of your system and whether you are running the installer from a local disk or a network share, you might get following security warning. Simply proceed clicking Run.

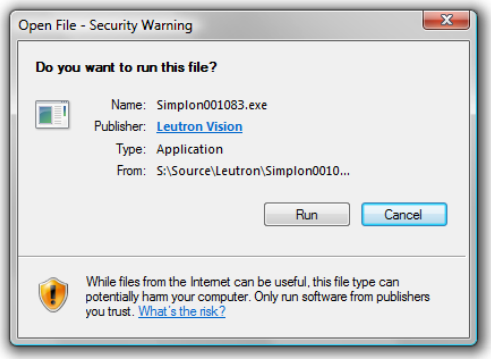

Open File - Security Warning
Do you want to run this file?
Name: Simplon001083.exe
Publisher: Leutron Vision
Type: Application
From: S:\Source\Leutron\Simplon0010...
Run
Cancel
While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Figure 4.13. Simplon installation: Windows security warning

4.5.1.3. Installation procedure

The paragraphs below discuss all the dialogs of Simplon's Windows installer and corresponding options. Note that some details might slightly differ between individual Simplon versions, but the principles will always be the same.

[2] http://www.leutron.com/support-downloads/download-area/

The installer first displays a welcome screen with basic information about the software you are going to install. Click Next to proceed.

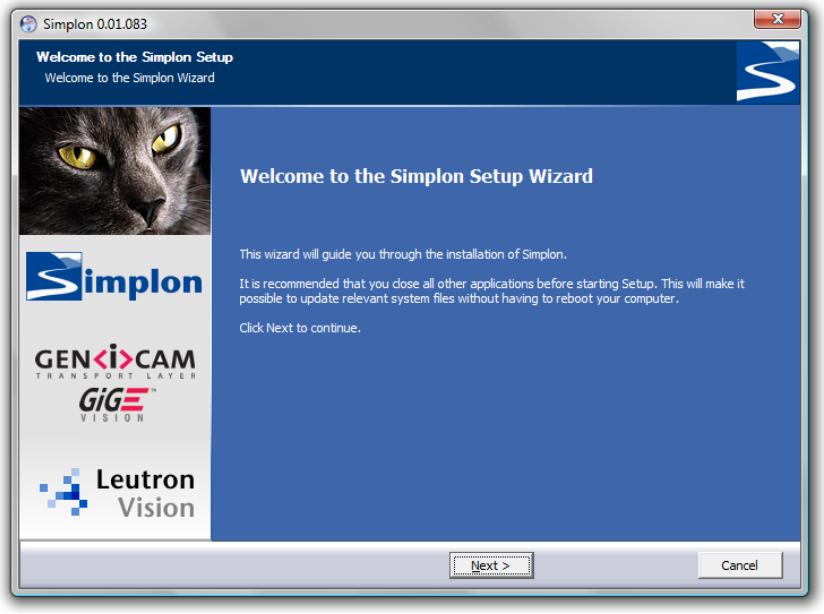

Simplon 0.01.083
Welcome to the Simplon Setup
Welcome to the Simplon Wizard
Welcome to the Simplon Setup Wizard
This wizard will guide you through the installation of Simplon.
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Simplon
GEN<i>CAM
TRANSPORT LAYER
GiGE VISION
Leutron Vision
Next >
Cancel

Figure 4.14. Simplon installation: welcome screen

Next screen is the license agreement. Please read carefully the license text and if you agree, click the I Agree button to proceed with installation. Once confirmed, your full agreement with the license text is assumed.

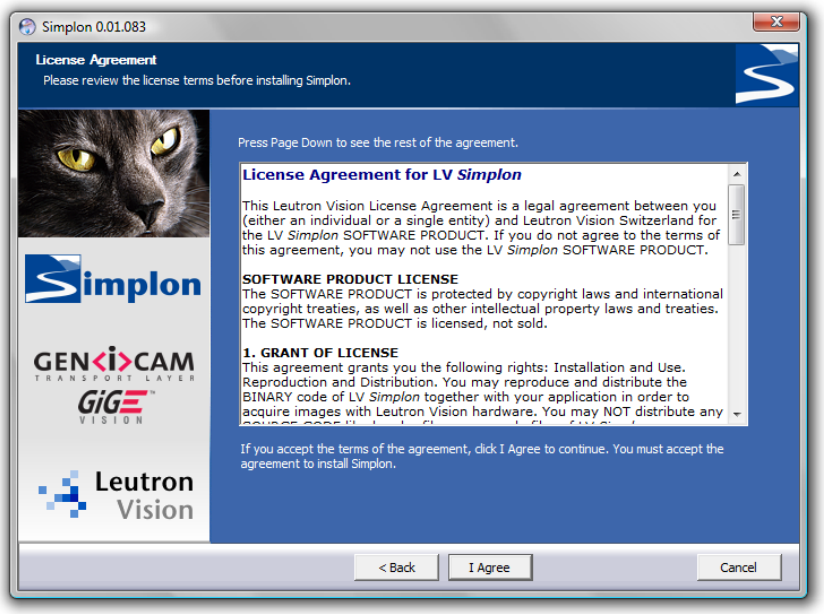

Simplon 0.01.083
License Agreement
Please review the license terms before installing Simplon.
Press Page Down to see the rest of the agreement.
License Agreement for LV Simplon
This Leutron Vision License Agreement is a legal agreement between you (either an individual or a single entity) and Leutron Vision Switzerland for the LV Simplon SOFTWARE PRODUCT. If you do not agree to the terms of this agreement, you may not use the LV Simplon SOFTWARE PRODUCT.
SOFTWARE PRODUCT LICENSE
The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.
1. GRANT OF LICENSE
This agreement grants you the following rights: Installation and Use. Reproduction and Distribution. You may reproduce and distribute the BINARY code of LV Simplon together with your application in order to acquire images with Leutron Vision hardware. You may NOT distribute any
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Simplon.
Simplon
GEN<i>CAM
TRANSPORT LAYER
GiGE VISION
Leutron Vision
< Back
I Agree
Cancel

Figure 4.15. Simplon installation: license agreement

Next the installer asks you about the location where to install Simplon. The default location `C:\Program Files\Leutron Vision\Simplon` should do well in most cases. If you prefer to install Simplon in another directory, select it now. When finished, click the Next button.

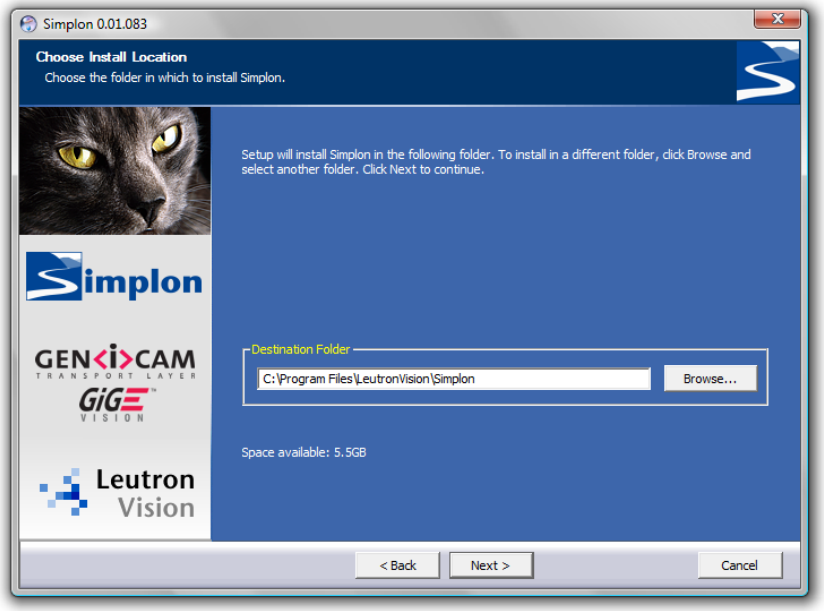


Figure 4.16. Simplon installation: destination directory

If running Windows Vista or newer Windows version, the default software installation directory (`C:\Program Files`) is read only for regular applications. Simplon, however, requires write access for certain files during operation (Section 4.5.1.4, “Installation layout” [p. 58]). Therefore the installer will ask you where to put the application data requiring write access.

Note that in Windows versions older than Windows Vista (in particular in Windows XP) the `C:\Program Files` area is not protected and therefore all data can go to a single directory. Also, when installing to a non-standard location (outside `C:\Program Files`), the installer assumes the user chooses a writable area and installs everything to that directory. The dialog will be omitted in these cases.

You can choose from three options:

- `C:\Users\Public\LeutronVision\Simplon` — recommended, when Simplon will be used by multiple users.
- `C:\Users\`*`YourUserName`*`\AppDAta\Local\LeutronVision\Simplon` – recommended, when Simplon will be used by a single user (you).
- User configured directory — if none of the options above fit your requirements, you can specify another directory. Be sure the directory has write access for all applications and is not protected by UAC.

Select the preferred option and click Next.

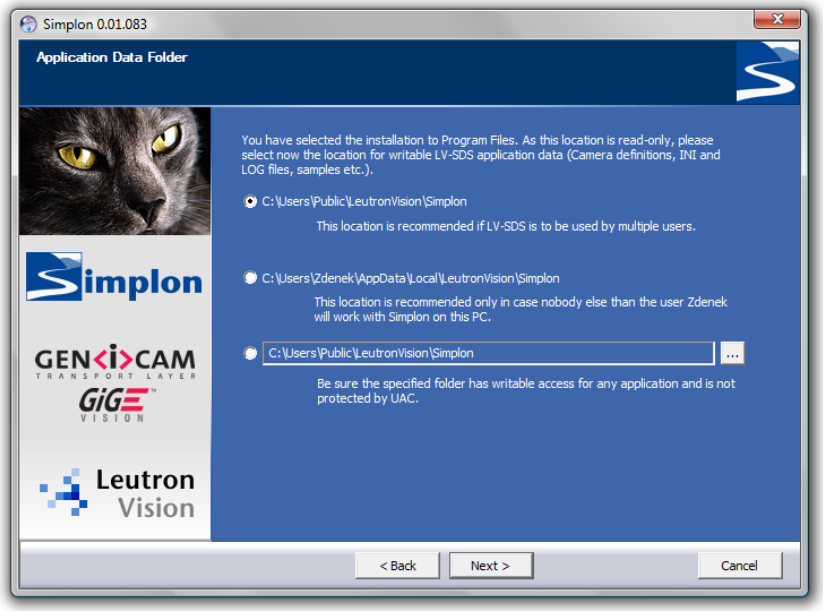

Simplon 0.01.083
Application Data Folder
You have selected the installation to Program Files. As this location is read-only, please select now the location for writable LV-SDS application data (Camera definitions, INI and LOG files, samples etc.).
C:\Users\Public\LeutronVision\Simplon
This location is recommended if LV-SDS is to be used by multiple users.
C:\Users\Zdenek\AppData\Local\LeutronVision\Simplon
This location is recommended only in case nobody else than the user Zdenek will work with Simplon on this PC.
C:\Users\Public\LeutronVision\Simplon
Be sure the specified folder has writable access for any application and is not protected by UAC.
Simplon
GEN<i>CAM
TRANSPORT LAYER
GiGE VISION
Leutron Vision
< Back
Next >
Cancel

Figure 4.17. Simplon installation: application data directory

In the next step you can select the Simplon components to install. The set of actual options can slightly differ among individual Simplon versions. The basic options are:

- Runtime files — files necessary to run every Simplon application, these are always installed.
- Development files — essential files for compiling against Simplon API as well as additional helper files. Files in this category are typically installed only on a development system, but not deployed with final runtime systems.
  - Include and library files — required for building Simplon API applications.
  - Sample code
- Documentation — full documentation set for Simplon package and all Leutron Vision camera families.

Note that for testing the cameras, the “runtime” components suffice. However, for programming with Simplon you need to make sure the Developer's tools are installed as well. When finished with the selection, click Install.

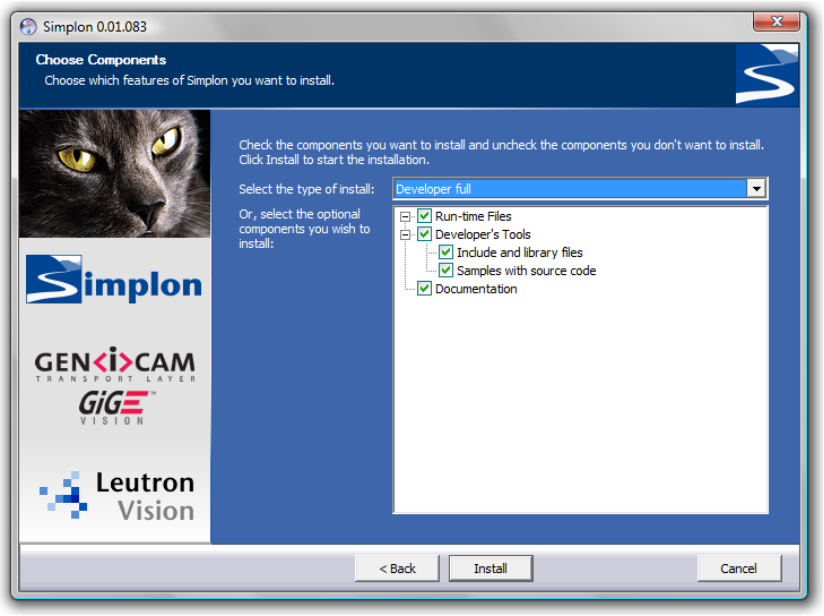


Figure 4.18. Simplon installation: selecting components

After starting the installation, the selected components will be installed to your computer. The installer will inform you about the progress. Note that the installation includes also Microsoft Visual Studio runtime redistributable, device driver for CheckSight cameras and network filter driver for GigE Vision based cameras (PicSight-GigE/PicSight-Smart).

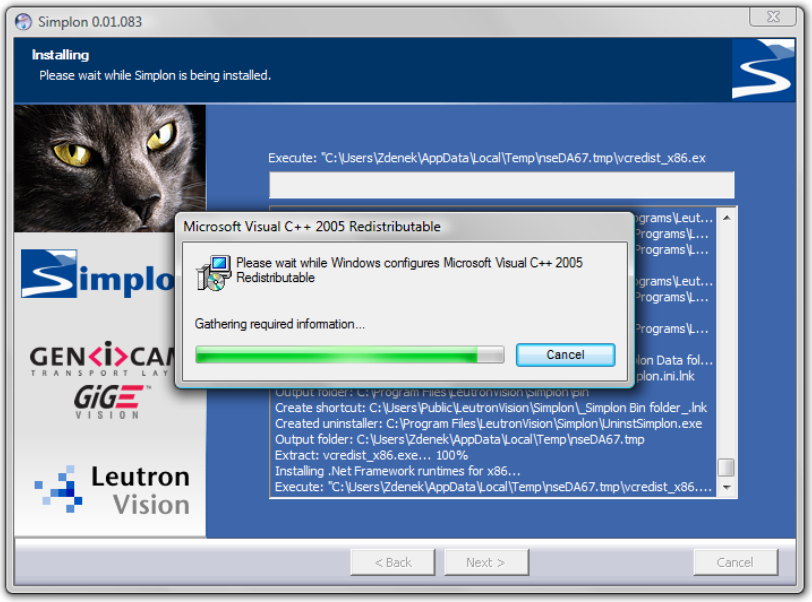


Figure 4.19. Simplon installation: installation progress

When completed, the installer displays the final dialog. We recommend to keep the default option (Reboot now) and click Finish. After rebooting, Simplon is ready to use.

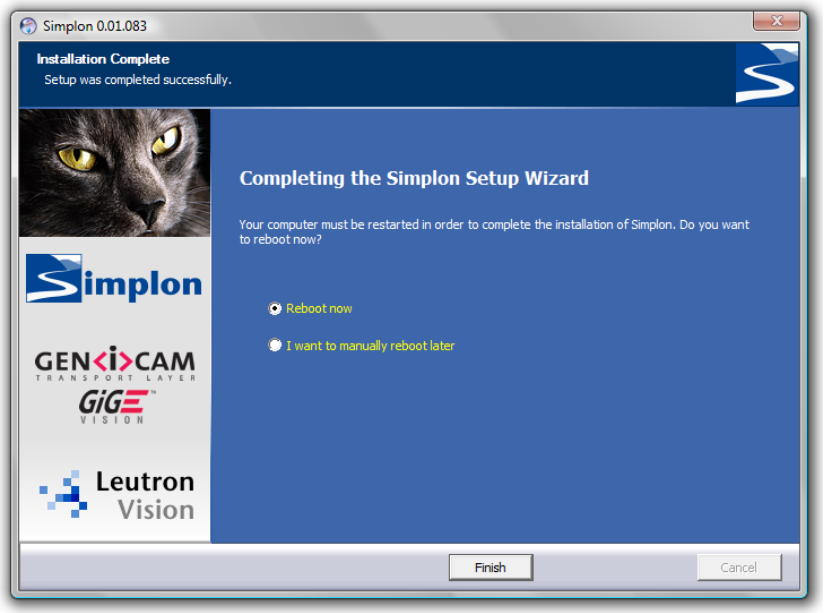


Figure 4.20. Simplon installation: finising installation

4.5.1.4. Installation layout

As mentioned above, the Simplon files are installed to two destinations, the main directory and the application data directory. For the older Windows versions (Windows XP), these two are merged and the application data directory is actually equivalent to the `bin` subdirectory of the main installation destination.

Main installation directory

The main installation directory contains the following subdirectories:

- `backup` — backup of user modified files (such as the configuration files) that might be overwritten when upgrading to newer Simplon version
- `bin` — all executable (`*.exe`) and library (`*.dll` files belonging to Simplon
- `data` — tool-specific data, eg. templates used by the Simplon Source Code Generator
- `doc` — documentation files
- `drivers` — drivers (CheckSight device driver, network filter driver)
- `include` — header files for Simplon API development
- `lib` — library (`*.lib`) files for Simplon API development
- `samples` — additional sample code (most of the samples can be, however, generated interactively using the Simplon Source Code Generator)

Application data directory

The application data directory contains following main components:

- `lv.simplon.ini` — Simplon configuration file.
- `lv.simplon.log` — Simplon log file.
- `XML` — contains static XML files copied during installation. None of the files in this directory should be removed unless being explicitly adviced by our support team.
- `XMLCache` — contains GenICam XML files downloaded from cameras.
- `Cache` — preprocessed versions of the GenICam XML files for performance improvement. These can be safely removed, Simplon will re-generate them upon next use.
- `Debug` — the `.pdb` files with debugging information for the Simplon libraries. These might be useful for troubleshooting when contacting our support team.

4.5.2. Installation in Linux

This section discusses Simplon package installation under Linux with notes specific for the most popular distributions and overview of system requirements.

4.5.2.1. Installation package types

Linux world does not have a single standard way for software installation. The Simplon Linux package is therefore distributed in multiple forms to suit all systems.

RPM format

- Used on most commercial distributions and their community variants, including Fedora/Red Hat, OpenSUSE/SUSE or Mandriva.
- The package file name is `simplon-x.yy.zzz.rpm`, where *x.yy.zzz* stands for Simplon version.
- To install, execute `rpm -i simplon-x.yy.zzz.rpm` with superuser privileges.
- To uninstall, execute `rpm -e simplon` with superuser privileges.

DEB format

- Used on Debian based distributions, including Debian, Ubuntu (and variants) or Mint.
- The package file name is `simplon-x.yy.zzz.deb`, where *x.yy.zzz* stands for Simplon version.
- To install, execute `dpkg -i simplon-x.yy.zzz.deb` with superuser privileges.
- To uninstall, execute `dpkg -r simplon` with superuser privileges.

Compressed archive (TARGZ)

- The `.tar.gz` archive is intended for systems, where the rpm/deb formats cannot be used or when a customized installation is required for serious reasons. Details are provided in a separate section.

4.5.2.2. System requirements

Following list enumerates the most important requirements for installing and using Simplon. Note that the list might not be complete and is subject to change between Simplon versions.

- Kernel version: kernels of the 2.6 line. The oldest tested version to date is 2.6.18, Simplon is kept aligned with all the newest kernel version.
- Compiler: g++/gcc 4.1 and newer, libstdc++ v. 6 (GLIBCXX 3.4.5), pthreads implementation NPTL.
- When installing on CheckSight, the kernel driver needs to get compiled and configured, therefore the basic infrastructure for kernel development needs to be available, in particular the basic compiling tools (gcc and make) and kernel headers. The name of the package containing the kernel headers might differ among individual distributions (eg. `linux-headers`, `kernel-devel` or similar).

  Additional system requirements might apply for CheckSight systems, if in doubt, contact us to get full list of distributions supported by the CheckSight cameras.

Copyright © Leutron Vision

- LSB compatibility and xdg-utils are advantage, but they come by default on most of the usual distributions.
- To compile samples genrated by the Simplon Source Code Generator, one needs basic development tools (make, g++) plus packages for Xlib/Xt development.

4.5.2.3. Standard installation layout

The Simplon installation follows the Filesystem Hierarchy Standard. Its files are distributed in several directories.

Main installation direcotry

The main isntallation directory is `/opt/simplon`.

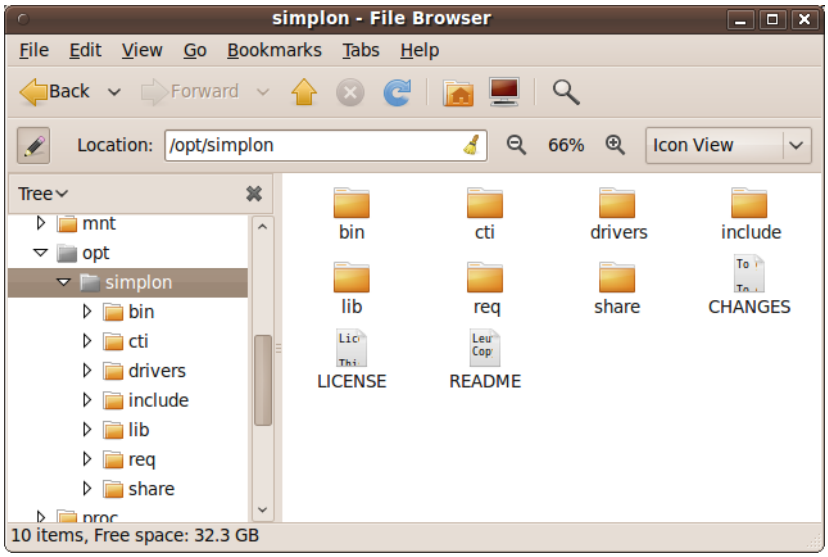


Figure 4.21. Linux installation layout, main directory

The main installation directory contains all the runtime binaries and development files. It is organized in following subdirectories:

- `bin` — tools and test/demo programs, in particular Simplon Explorer (`lv.explorer`) and Simplon Settings (`lv.settings`)
- `lib` — all Simplon API libraries, ie. a directory where the linker should be pointed at; on 64-bit systems the `lib` directory contains the 32-bit versions, while the 64-bit ones reside in `lib64`
- `cti` — contains the Simplon GenTL Producer library; on 64-bit systems the `cti` directory contains the 32-bit versions, while the 64-bit ones reside in `cti64`
- `include` — Simplon API header files
- `share` — static data files used by Simplon
- `share/xml` — static XML files copied during installation
- `req` — foreing requisite components used by Simplon, in particular the GenICam runtime
- `drivers` — scripts and sources necessary to build and install Simplon device drivers (used only on CheckSight systems)

Configuration files

The configuration files are stored in `/etc/opt/simplon`.

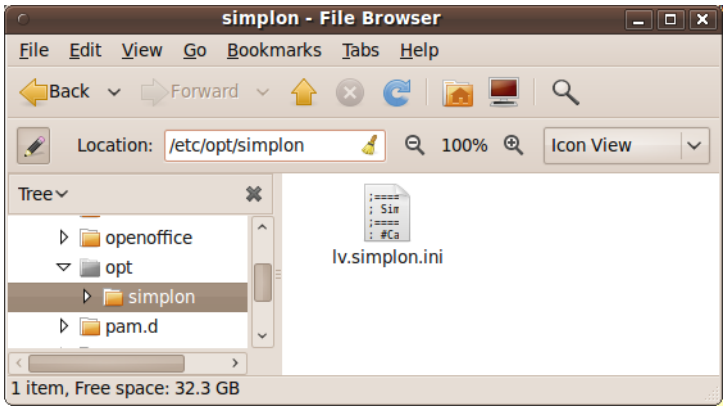


Figure 4.22. Linux installation layout, configuration files

The directory contains in particular the main Simplon configuration file, `lv.simplon.ini`. The file should be edited with superuser privileges. See Section 5.3.1, "Simplon Settings utility and the configuration file" [p. 76] for details.

Application data

The various application data, such as logs and cache go to `/var/opt/simplon`.

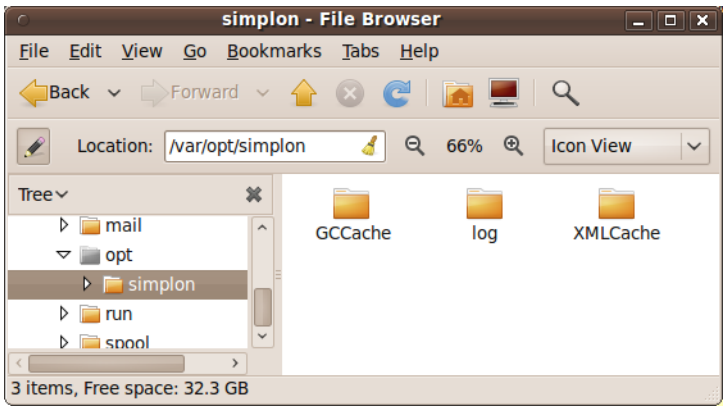


Figure 4.23. Linux installation layout, application data

The data are stored in several subdirectories, in particular:

- `log` — default destination for Simplon log files, in particular the `lv.simplon.log`, that are useful for any troubleshooting (refer to Chapter 9, *Troubleshooting & Support* [p. 234]).
- `XMLCache` — contains GenICam XML files downloaded from cameras.
- `GCCache` — preprocessed versions of the GenICam XML files for performance improvement. These can be safely removed, Simplon will re-generate them upon next use.

User specific files

Possible user specific files (such as demos'/tools' configuration) are stored in `~/.simplon`.

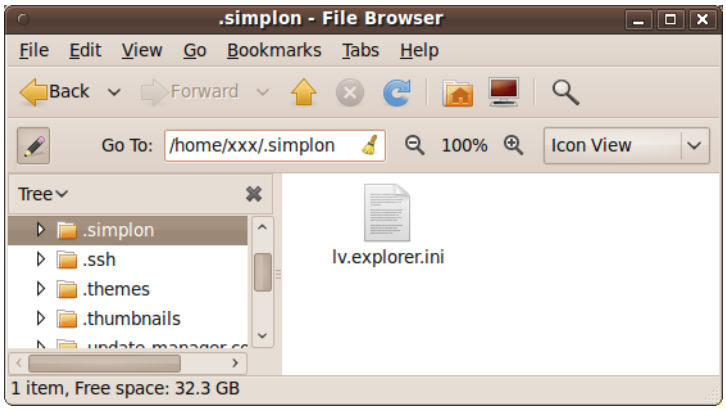


Figure 4.24. Linux installation layout, user specific data

This directory is used solely to store user specific settings or preferences for Simplon tools, such as Simplon Explorer or Simplon Settings. It will not be needed for usual operation on a runtime system.

Other actions performed by the installer

Besides copying the files and adjusting user rights, the installer performs some additional actions, including:

- When installing on a CheckSight system (detected automatically), the necessary device driver is built and installed.
- For convenience, the directory `/opt/simplon/bin` is added to `PATH` environment variable, so that the tools are directly accessible from the shell. This is performed by adding an entry to `/etc/profile.d` — on systems using other mechanism, the system administrator might need to adjust the path manually.
- Directory `/opt/simplon/cti` is added to the `GENICAM_GENTL32_PATH` environment variable (and analogically the 64-bit variant), so that the Simplon GenTL Producer can be located by 3rd party GenTL consumers. This is performed by adding an entry to `/etc/profile.d` — on systems using other mechanism, the system administrator might need to adjust the path manually.
- For convenience, the directory `/opt/simplon/lib` (and it's 64-bit variant) is added to the search path of the dynamic linker. This is performed by adding an entry to `ld.so.conf.d` — on systems using other mechanism, the system administrator might need to adjust the path manually.

Custom installations

Installation to other than default locations is possible, however we recommend to do that only for well-legitimate reasons. Corresponding guidelines are provided in a separate section.

4.5.2.4. Notes for the most common distributions

The Simplon package is tested with the most popular distributions. Brief notes for those distributions are listed below. Note that this list by no means suggests that other distributions are not supported, Simplon should work on any system fulfilling the basic requirements (Section 4.5.2.2, “System requirements” [p. 59]). Also the versions listed for each distribution are just a hint, showing which was the original version used for tests. Each newer version of given distribution should work well and older versions would work if fulfilling the mentioned system requirments.

For each distribution, the tests were done with the most basic “default” setup including X, but without any special additions.

The notes below are related only to Simplon itself, assuming the Linux operating system was properly installed and configured. Special requirements related to operating system installation might apply on CheckSight systems —. To install Simplon on CheckSight, kernel development infrastructure (gcc, make, kernel headers) must be installed to succesfully build and configure the device driver.

Ubuntu

- Originally tested version: 9.10 (Karmic Koala)
- Packaging type: deb format
- Works out of the box

Debian

- Originally tested version: 5.0 (Lenny)
- Packaging type: deb format
- Works out of the box

OpenSUSE (community version of SUSE Linux Enterprise)

- Originally tested version: 11.2
- Packaging type: rpm format
- Default installation switches on a paranoid firewall, which has to be configured (or switched off) if working with GigE Vision cameras (PicSight-GigE/PicSight-Smart)

Fedora (community version of Red Hat Enterprise Linux)

- Originally tested version: 12 (Constantine)
- Packaging type: rpm format
- Default installation switches on a paranoid firewall, which has to be configured (or switched off) if working with GigE Vision cameras (PicSight-GigE/PicSight-Smart)

Mandriva

- Originally tested version: 2010.0
- Packaging type: rpm format
- Default installation switches on a paranoid firewall, which has to be configured (or switched off) if working with GigE Vision cameras (PicSight-GigE/PicSight-Smart)

Slackware

- Originally tested version: 13
- Packaging type: special (tgz based), but slackware has a tool rpm2tgz which can be used to convert the rpm package (`rpm2tgz simlpon-x.yy.zzz.rpm`) and then install it (`installpkg simplon-x.yy.zzz.tgz`)
- Works out of the box

Gentoo

- Originally tested version: 10.1
- Packaging type: special, no reasonable way to use the rpm/deb packages, need to use the tar.gz package

Mint

- Originally tested version: 8 (Helena)
- Packaging type: deb format
- Works out of the box

Copyright © Leutron Vision

5. Initial configuration and preparation

5.1. BIOS

5.1.1. BIOS specifications

The CheckSight system uses an AMI BIOS with following specifications:

Embedded BIOS	Description
ACPI PM	V3.0 Power Management
ACPI Battery Support	Not enabled
Boot Devices	PATA, USB-Devices, LAN
CMOS-Data	Battery backed SRAM and a copy in the EEPROM
BIOS Setup Backup	Automatically written into the EEPROM after setup-screen exit
Battery-less Boot	Generally possible, setup data are transferred from the EEPROM. Without backup-battery, the time and date counter is not running while the computer system is switched off.
Customized CMOS	Possible for OEMs, must be ordered separately
Customized User Data	Possible for OEMs, must be ordered separately
PCI-Express	Supports PCI V2.2
IDE Mode	Compatible mode (IRQ14/15) or native mode (PCI device)
APIC-Support	Yes, Advanced Programmable Interrupt Controller
AHCI	No
EIST	Yes, Enhanced Intel SpeedStep Technology
Intel VT	Yes
Thermal Management	Yes, integrated in the ACPI. • Throttling processor function • Catastrophic temperature (hardware shutdown)
Supported ACPI Suspend Modes	• S1 (POS = Power On Suspend) • S3 (STR = Suspend to RAM)

Table 5.1. CheckSight BIOS specifications

5.1.2. BIOS Setup configuration

To enter the BIOS Setup utility, press the **Delete** key when the CheckSight starts.

To enter the boot menu (boot device selection), press the **F11** key when the CheckSight starts.

The following figures demonstrate the configuration options of the CheckSight camera's BIOS Setup utility. The values shown at the screenshots are default values of the CheckSight BIOS settings. The BIOS Setup always displays brief documentation for the currently selected option.

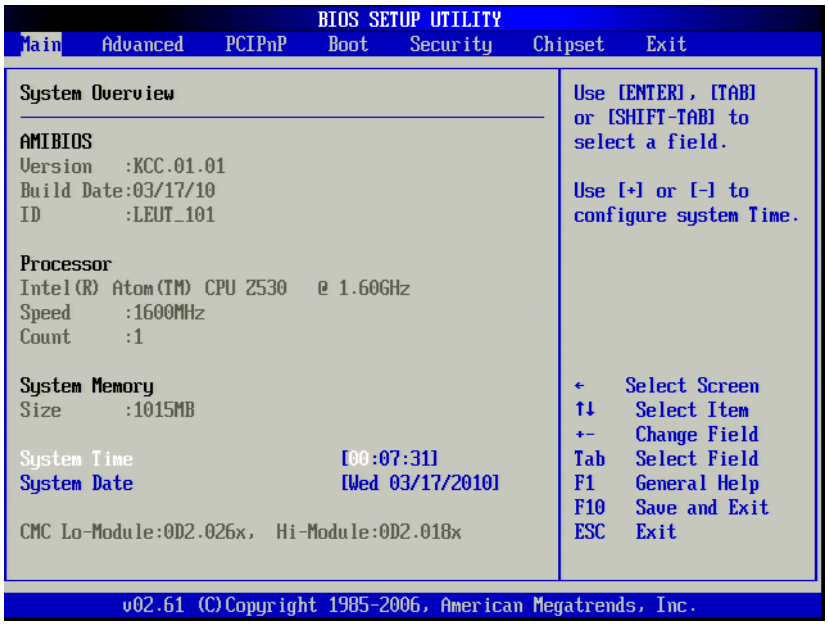


Figure 5.1. BIOS Setup, main menu

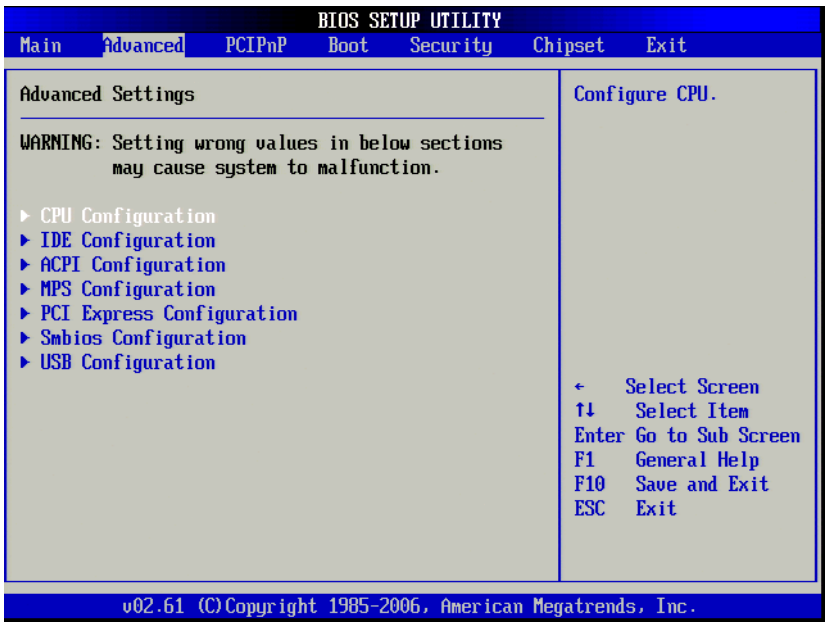


Figure 5.2. BIOS Setup, advanced settings

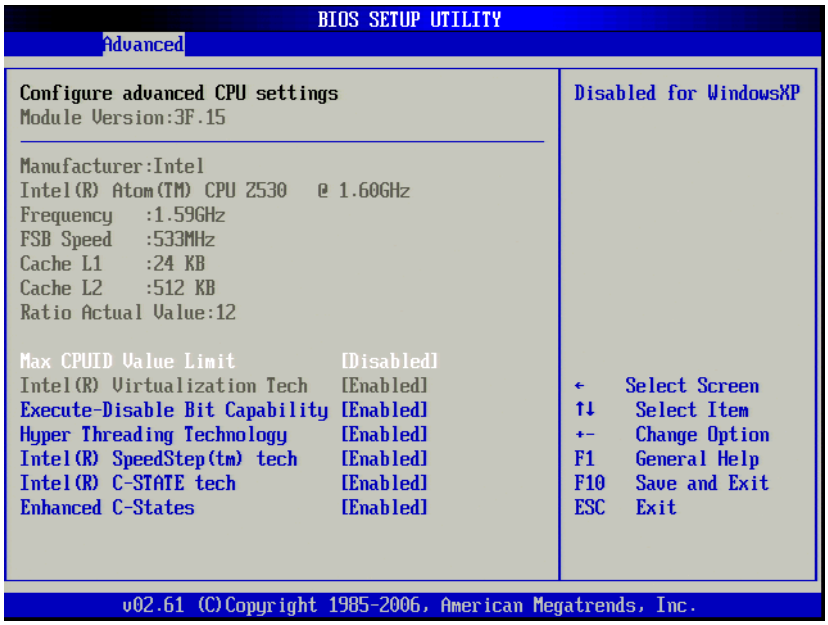


Figure 5.3. BIOS Setup, advanced CPU configuration

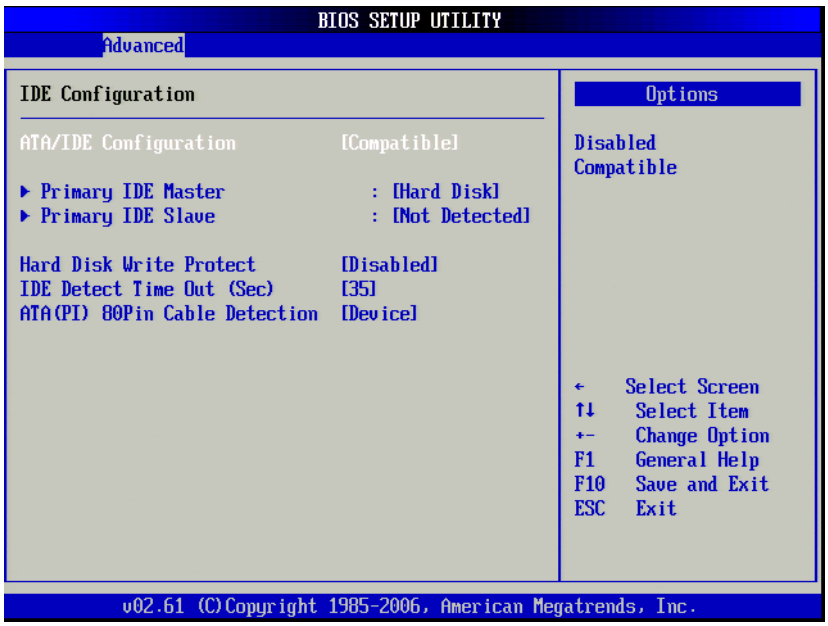


Figure 5.4. BIOS Setup, advanced IDE configuration

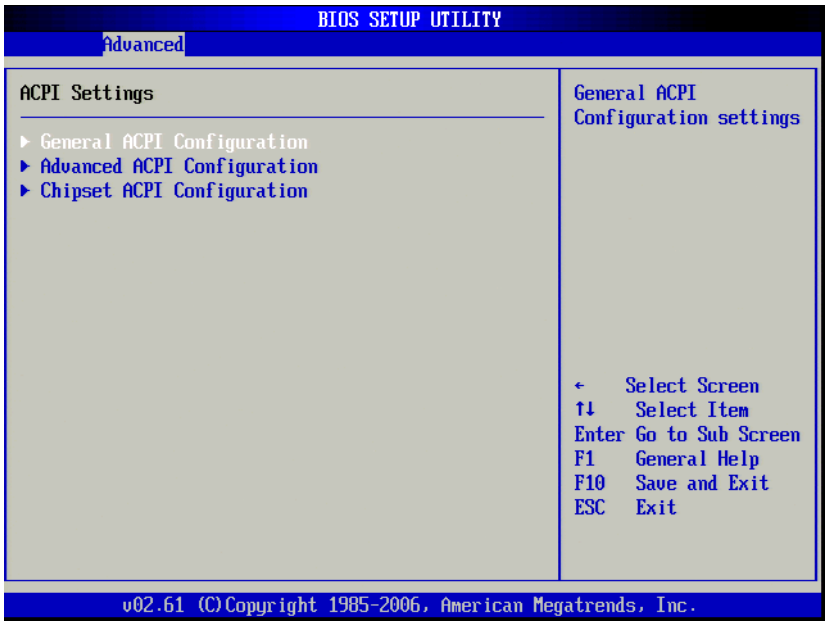


Figure 5.5. BIOS Setup, advanced ACPI configuration

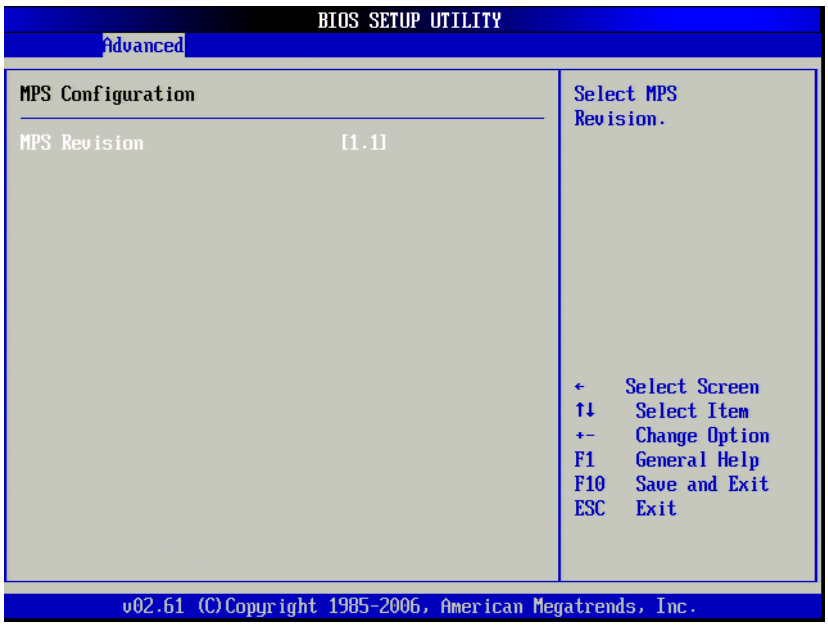


Figure 5.6. BIOS Setup, advanced MPS configuration

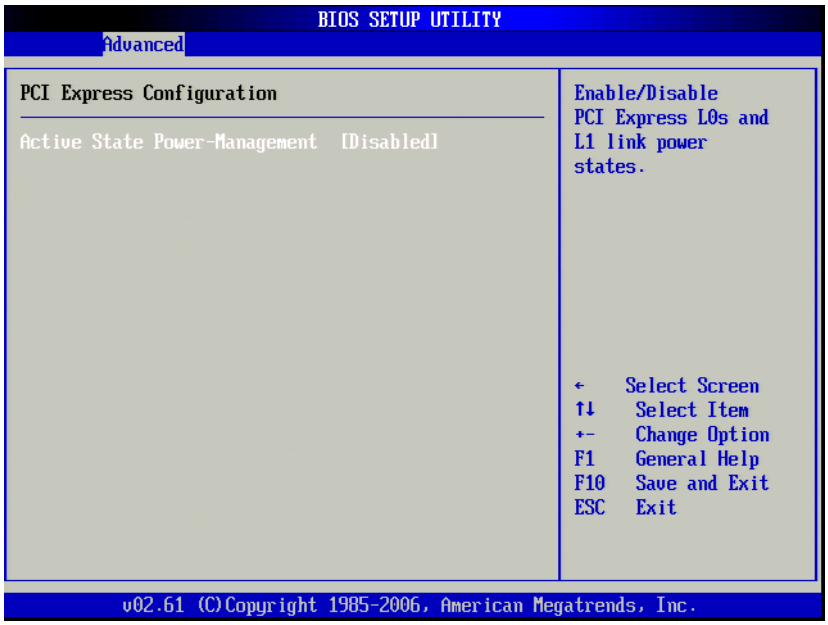


Figure 5.7. BIOS Setup, advanced PCI-Express configuration

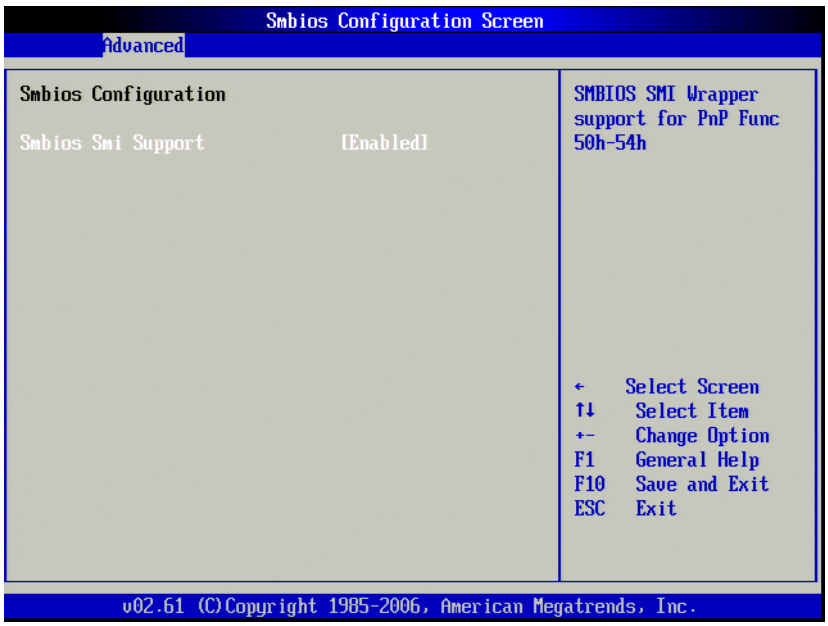


Figure 5.8. BIOS Setup, advanced SMBIOS configuration

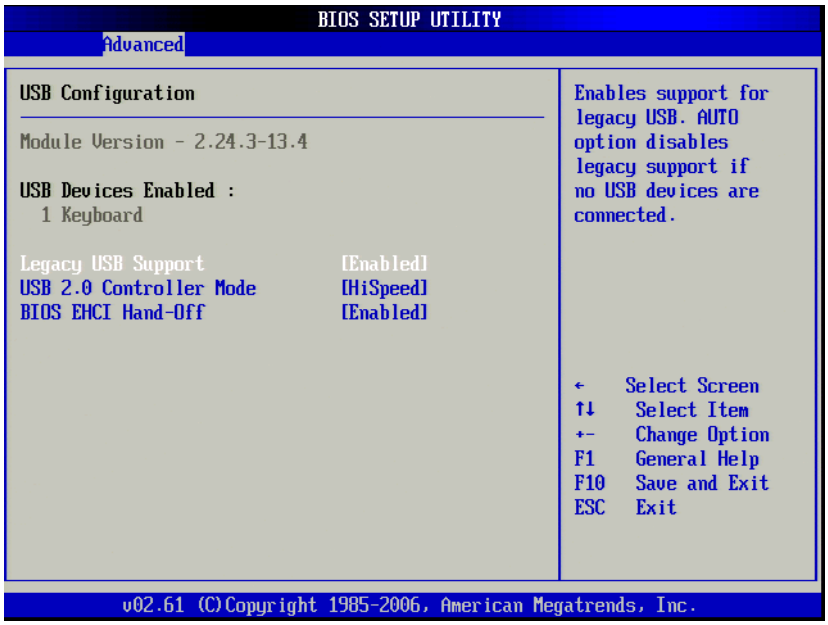

BIOS SETUP UTILITY
Advanced
USB Configuration
Module Version - 2.24.3-13.4
USB Devices Enabled :
1 Keyboard
Legacy USB Support [Enabled]
USB 2.0 Controller Mode [HiSpeed]
BIOS EHCI Hand-Off [Enabled]
Enables support for legacy USB. AUTO option disables legacy support if no USB devices are connected.
← Select Screen
↑↓ Select Item
+- Change Option
F1 General Help
F10 Save and Exit
ESC Exit
v02.61 (C)Copyright 1985-2006, American Megatrends, Inc.

Figure 5.9. BIOS Setup, advanced USB configuration

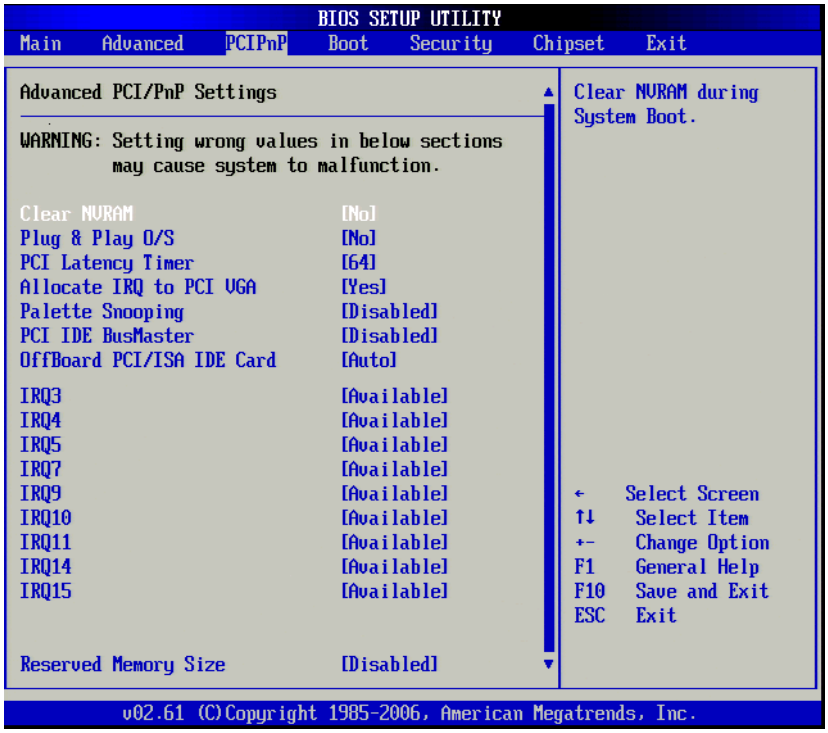


Figure 5.10. BIOS Setup, advanced PCI/PnP settings

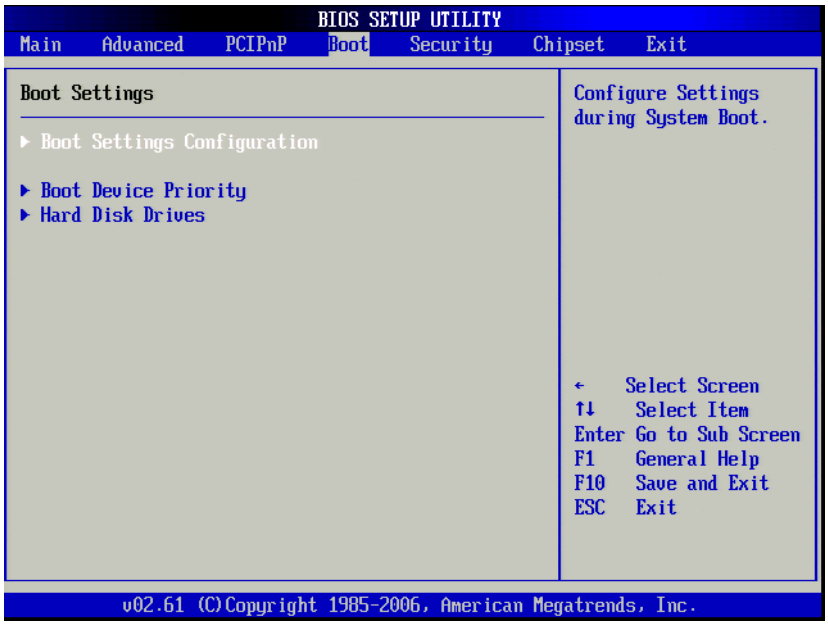


Figure 5.11. BIOS Setup, boot settings

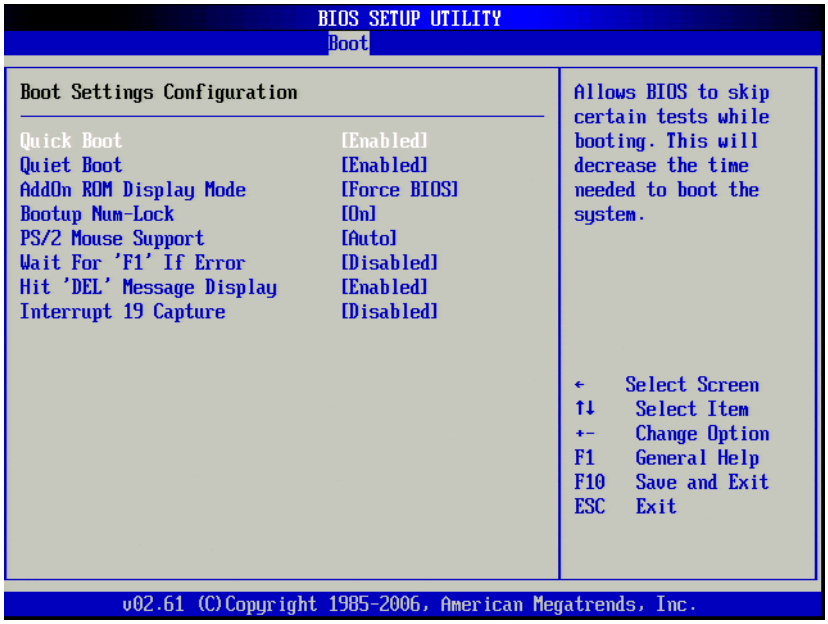

BIOS SETUP UTILITY
Boot
Boot Settings Configuration
Quick Boot [Enabled]
Quiet Boot [Enabled]
AddOn ROM Display Mode [Force BIOS]
Bootup Num-Lock [On]
PS/2 Mouse Support [Auto]
Wait For 'F1' If Error [Disabled]
Hit 'DEL' Message Display [Enabled]
Interrupt 19 Capture [Disabled]
Allows BIOS to skip certain tests while booting. This will decrease the time needed to boot the system.
← Select Screen
↑↓ Select Item
+- Change Option
F1 General Help
F10 Save and Exit
ESC Exit
v02.61 (C)Copyright 1985-2006, American Megatrends, Inc.

Figure 5.12. BIOS Setup, boot settings configuration

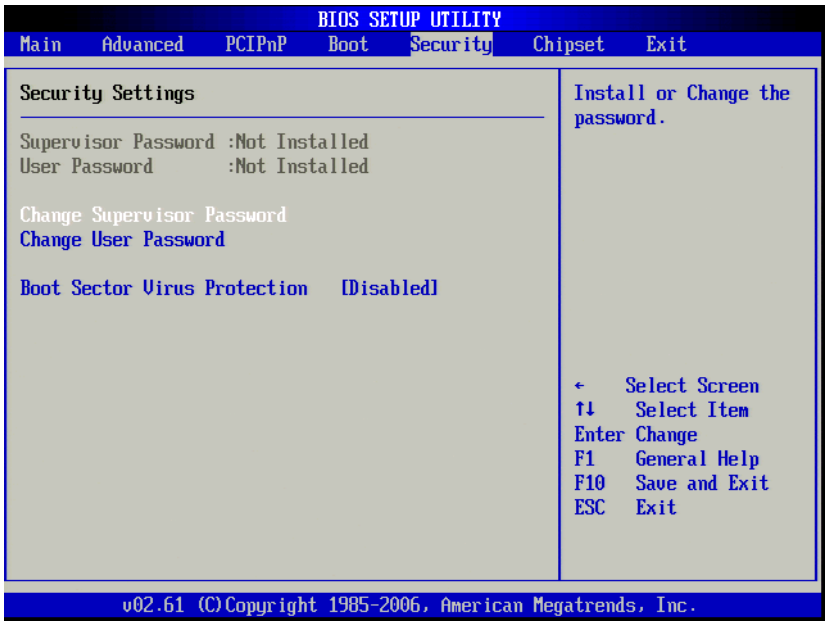


Figure 5.13. BIOS Setup, security settings

BIOS SETUP UTILITY
Main Advanced PCIPnP Boot Security Chipset Exit
Advanced Chipset Settings
WARNING: Setting wrong values in below sections may cause system to malfunction.
North Bridge Configuration
South Bridge Configuration
Configure North Bridge features.
← Select Screen
↑↓ Select Item
Enter Go to Sub Screen
F1 General Help
F10 Save and Exit
ESC Exit
v02.61 (C)Copyright 1985-2006, American Megatrends, Inc.

Figure 5.14. BIOS Setup, chipset settings

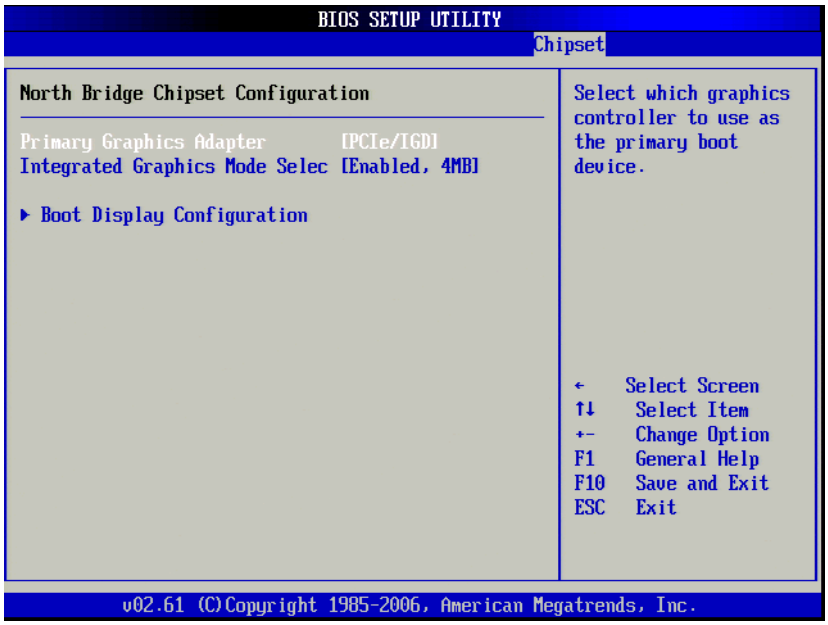

BIOS SETUP UTILITY
Chipset
North Bridge Chipset Configuration
Primary Graphics Adapter [PCIe/IGD]
Integrated Graphics Mode Selec [Enabled, 4MB]
▸ Boot Display Configuration
Select which graphics controller to use as the primary boot device.
← Select Screen
↑↓ Select Item
+- Change Option
F1 General Help
F10 Save and Exit
ESC Exit
v02.61 (C)Copyright 1985-2006, American Megatrends, Inc.

Figure 5.15. BIOS Setup, chipset northbridge configuration

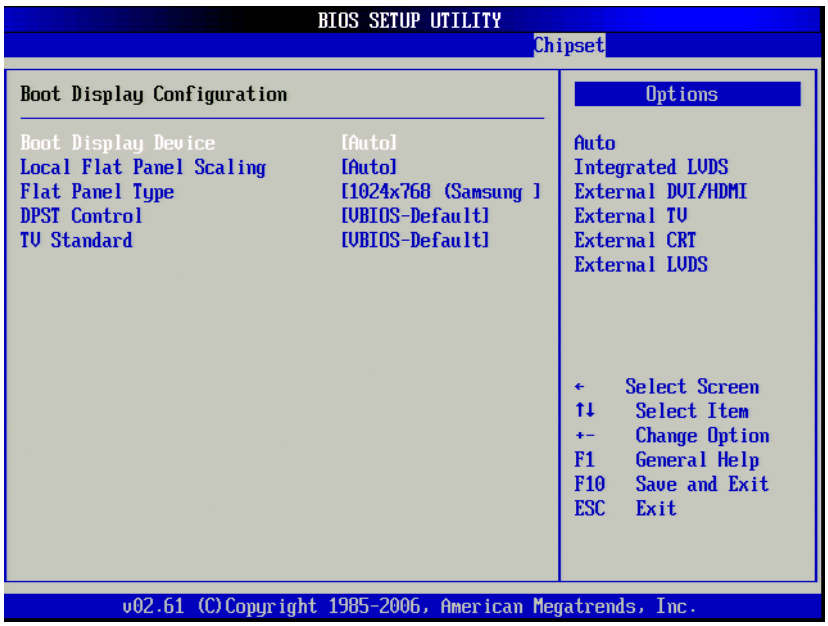


Figure 5.16. BIOS Setup, chipset northbridge display

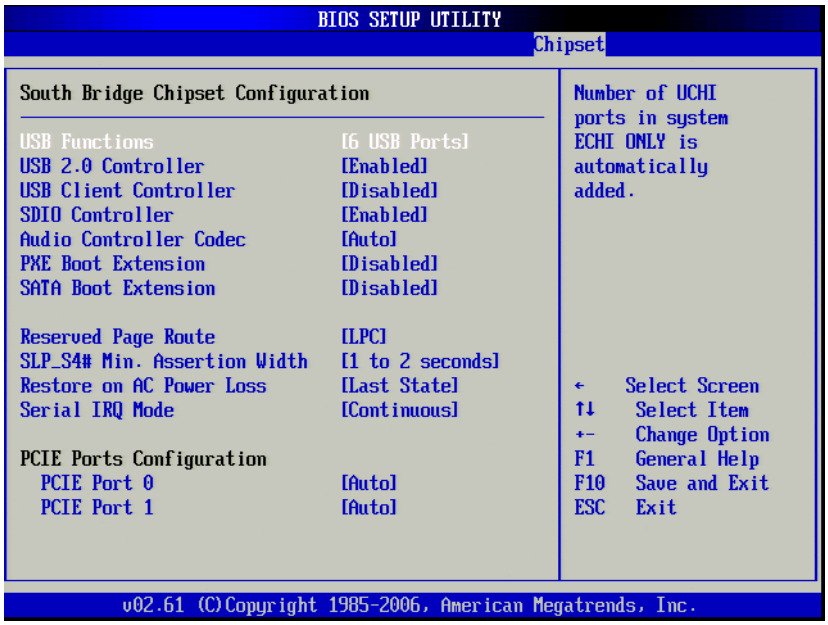

BIOS SETUP UTILITY
Chipset
South Bridge Chipset Configuration
USB Functions [6 USB Ports]
USB 2.0 Controller [Enabled]
USB Client Controller [Disabled]
SDIO Controller [Enabled]
Audio Controller Codec [Auto]
PXE Boot Extension [Disabled]
SATA Boot Extension [Disabled]
Reserved Page Route [LPC]
SLP_S4# Min. Assertion Width [1 to 2 seconds]
Restore on AC Power Loss [Last State]
Serial IRQ Mode [Continuous]
PCIE Ports Configuration
PCIE Port 0 [Auto]
PCIE Port 1 [Auto]
Number of UCHI ports in system ECHI ONLY is automatically added.
← Select Screen
↑↓ Select Item
+- Change Option
F1 General Help
F10 Save and Exit
ESC Exit
v02.61 (C)Copyright 1985-2006, American Megatrends, Inc.

Figure 5.17. BIOS Setup, chipset southbridge configuration

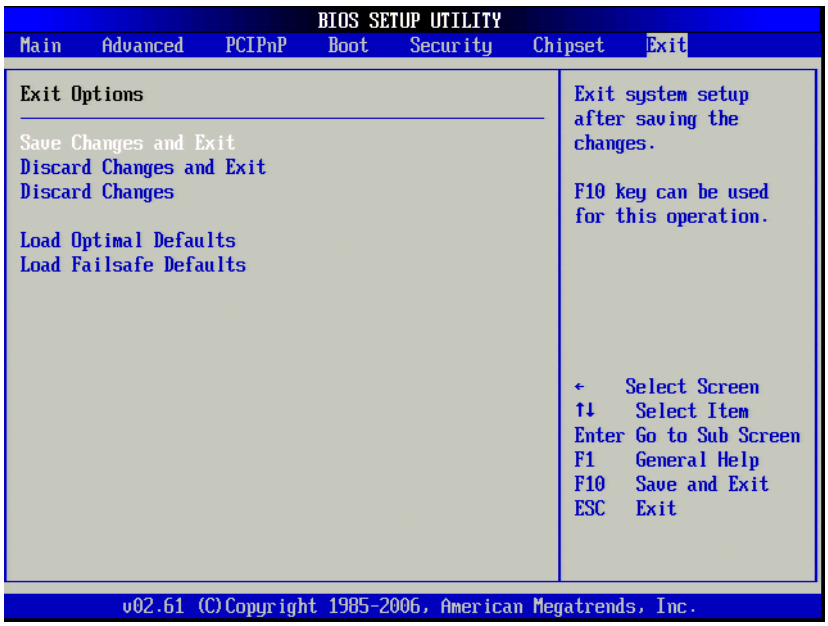

BIOS SETUP UTILITY
Main Advanced PCIPnP Boot Security Chipset Exit
Exit Options
Save Changes and Exit
Discard Changes and Exit
Discard Changes
Load Optimal Defaults
Load Failsafe Defaults
Exit system setup after saving the changes.
F10 key can be used for this operation.
← Select Screen
↑↓ Select Item
Enter Go to Sub Screen
F1 General Help
F10 Save and Exit
ESC Exit
v02.61 (C)Copyright 1985-2006, American Megatrends, Inc.

Figure 5.18. BIOS Setup, exit

5.1.3. BIOS upgrade

Before upgrading the BIOS, please check the following:

- Make a bootable USB device or Compact Flash which includes the following files:
  - `kccflash.exe`
  - `dldelep.exe`
  - BIOS upgrade: `bios.rom`
  - `update.bat`
- If necessary rename the BIOS upgrade to `bios.rom`.
- Content of `update.bat`:

- `dldelep /y`
- `kccflash -f bios.rom`

- Disable the EMM386 or other memory managers in the `config.sys` of your boot device.
- Make sure that the `kccflash.exe`, the `dldelep.exe` program and the `bios.rom` to be used for upgrade are in the same path and directory.
- To Boot DOS without `config.sys` and `autoexec.bat`, press **F5** while starting the DOS boot.
- Is the empty drive space, where the `kccflash.exe` is located, larger than 1024 kB (for backup `kccflash.exe -s` or `-r`)?
- Isn't the drive write-protected?

The BIOS upgrade procedure:

1. Start the system with the bootable Compact Flash or USB device. If you do not have modified your `autoexec.bat` and `config.sys` you can press the **F5** key to disable all the entries and start with DOS prompt.
2. Run `update.bat`. `dldelep.exe /Y` will clear the CMOS and the EEPROM. The `/Y` is for the confirmation, if you don't use this `/Y` you must confirm manually by pressing **Y** (warning: on some keyboards **Z** and **Y** keys might be swapped). If you do not run `dldelep.exe /Y`, the system will be destroyed during the BIOS upgrade!
3. `kccflash.exe /f bios.rom` will upload the `bios.rom` file to the system. The `kccflash.exe` utility recognizes following parameters:
   - `-f` *`filename`* — filename of the new BIOS file
   - `-s` *`filename`* — filename for backup of current BIOS
   - `-r` *`filename`* — backup current BIOS to a file with specified filename and exit the program
   - `-y` — say "yes" to all questions
   - `-i` — list all supported chipsets and flash chips
4. When the BIOS upgrade is finished, you must power off the system.
5. After powering the system back on, press **Delete** to enter the setup mode and set the default values pressing **F9**.
6. "Save and leave" the setup pressing **F10**.
7. The system will reboot. The upgrade procedure is finished.

Do not use boot disks created in a Windows operating system. If you do not have an MS-DOS 6.22 disk available, you can download a boot disk from www.boot-disk.com.

5.2. Firmware update

The CheckSight cameras are field upgradeable. The upgrade procedure is simple and can be done using the CheckSight Firmware Update Tool tool.

The following description assumes that you downloaded the package containing the new firmware from our download area[1] or that you obtained the package from our support staff. The package file name comes in shape `F`*`xxxx`*`_R`*`nnnn`*`.xsvf`, where *`xxxx`* is a set of letters, digits and underscores defining the CheckSight type the update package is intended for and *`nnnn`* stands for

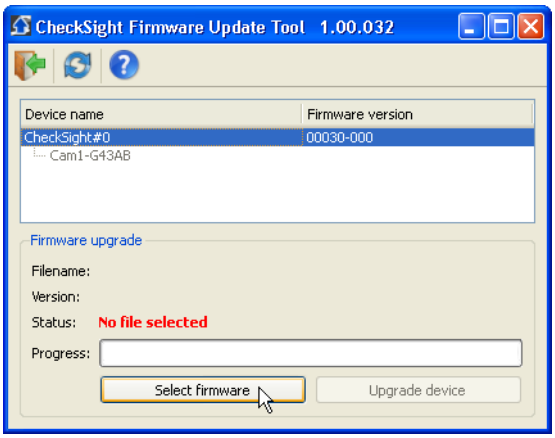
[1] http://www.leutron.com/support-downloads/download-area/

firmware version, eg. *00030*). The upgrade tool will recognize the package type suitable for given CheckSight from the filename pattern.

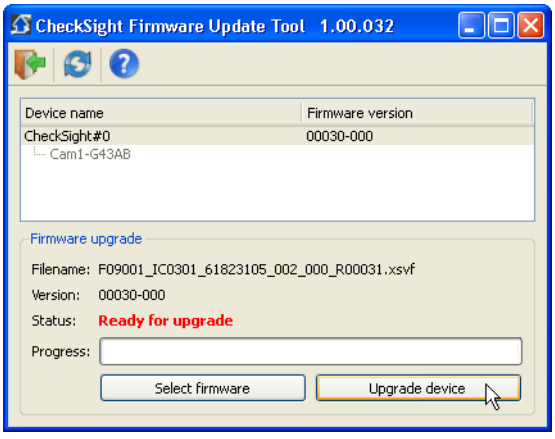
Now the update procedure itself (note that the exact look of the CheckSight Firmware Update Tool might slightly vary among individual Simplon versions):

1. Start CheckSight Firmware Update Tool. When run on a CheckSight system, it will display the actual CheckSight model, including other information, in particular version of the firmware currently loaded on the camera. Select the line with CheckSight description and click the Select firmware button to select the firmware update package. The tool will ask you to browse for the update package to be used — locate it and confirm.

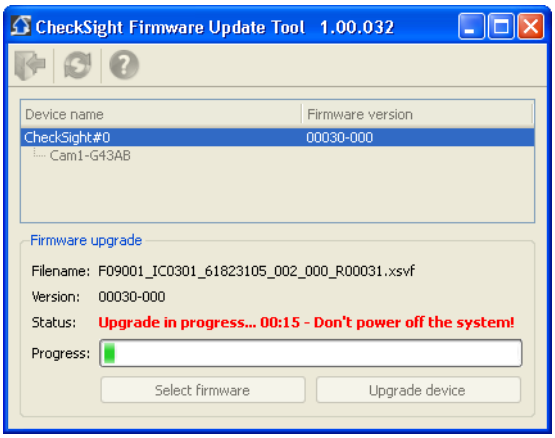
   When running the tool under Linux, we recommend to execute it with superuser privileges — so that it does not have any problems with the automatic shutdown in the last step of the update procedure.



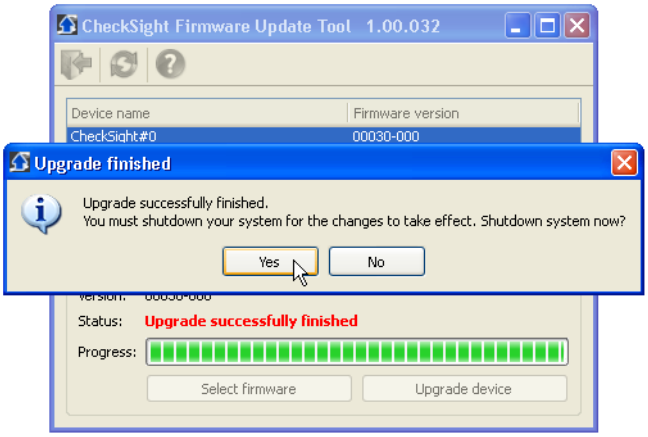
2. When selected, click the Upgrade device button to start the actual update procedure. Note that the camera must not be shut down or powered off during the update procedure, otherwise the camera might get corrupted.



3. The tool will display progress of the operation, with warning that the system must not be powered off.



4. When finished, the CheckSight needs to be powered off (real shutdown, not just a restart) to get the new firmware applied. The tool will attempt to shutdown the system automatically after your confirmation. If the system is not powered off automatically, shutdown manually. After next boot, the new firmware will be loaded.



5.3. Simplon configuration

Following section provides basic information about Simplon software package configuration. More information about the Simplon configuration, troubleshooting and other related topics can be found in Simplon getting started.

5.3.1. Simplon Settings utility and the configuration file

The Simplon configuration options are concentrated in a single file, `lv.simplon.ini`. On Windows systems the file is located in the “application data” part of the installation. On Linux systems it's stored by default in `/etc/opt/simplon`.

The file can be edited directly — every single option is well documented directly in the file, including it's possible values. Note that on Linux superuser privileges are required to edit the file.

Instead of editing the file directly, however, the recommended alternative is to edit the file through a helper utility, Simplon Settings (Windows: menu Start → Programs → Leutron Vision Simplon → Tools → Simplon Settings; Linux: /opt/simplon/bin/lv.settings as superuser). The tool provides a comfortable way to edit the `lv.simplon.ini` file.

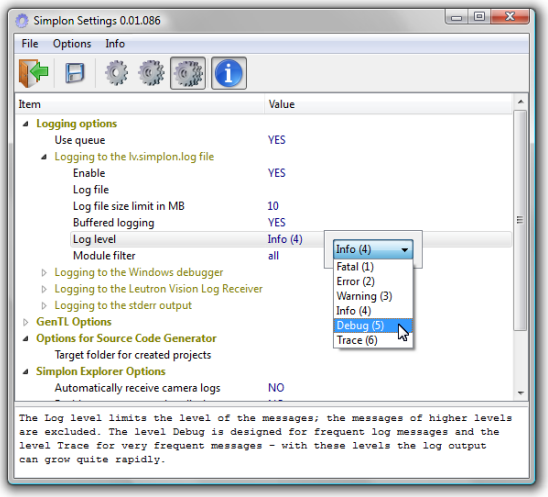


Figure 5.19. Simplon Settings utility

The tool's interface can be controlled threw a handful of menu options (or corresponding buttons). You can show/hide the info pane with info about the selected configuration option. You can switch between three configuration levels (beginner, expert, guru) — the higher level, the more detailed options are shown. Finally, you can store the changes.

The set of options itself is ever evolving, so the Simplon Settings tool itself (or the `lv.simplon.ini` file) is the most complete documentation. The options include detailed logging configuration (see also Chapter 9, *Troubleshooting & Support* [p. 234]), system installation related info and configuration of individual Simplon subsystems.

5.3.2. Selected configuration options

The documentation for all the Simplon configuration options is provided directly in the Simplon Settings utility and the configuration file. Here we'll mention just few selected important options:

Logging options

The Simplon log is an important tool for any support and troubleshooting. For regular operation, the log might be kept at low level (default is level `Info (4)`) or even switched off to optimize the performance. However, during application development or when debugging a problem, we recommend to switch the `Log level` configuration option to `Trace (6)`. All log files sent to our support team should be created with level `Trace (6)`.

The logging infrastructure is optimized for high performance. For example, the logging to file is buffered to minimize the disk access overhead. This can have a drawback, however, that in case of an application crash, the last part of the log might be missing in the file. When troubleshooting application crashes, we recommend to switch on the option `Disable all buffering` .

Heartbeat timeout

`Heartbeat timeout` is an important parameter, related to GigE Vision cameras only (PicSight-GigE/PicSight-Smart).

This timeout, in milliseconds, defines how frequently has the camera send the “heartbeat” messages to the camera, indicating that it's still alive. When the camera does not receive the message within the specified timeout period, it assumes the application has gone away and disconnects itself. The default value used in Simplon is 3 s.

At most times, the heartbeat handling is fully transparent to the user. However, during debugging, when a developer stops the application at a breakpoint for single stepping, the Simplon heartbeat handling thread would also stop, fail to deliver the heartbeat messages and the camera would disconnect. To prevent that, we recommend to increase the heartbeat timeout parameter to a higher value. The exact value should be a compromise between being able to debug comfortably, while understanding that when the application stops unexpectedly without disconnecting the camera properly, it will take a while before the camera disconnects automatically and becomes available for another connection.

The parameter is configurable through camera features at runtime, while its default value can be adjusted through the Simplon Settings utility.

5.4. Acquisition and runtime configuration

The camera runtime functionality is fully configurable through the GenICam interface. The individual features, their relationships and use is described in chapter Chapter 6, *Camera features and their control* [p. 79].

Accessing the camera (feature control and image acquisition) using the Leutron Vision Simplon API interface is documented in Simplon API documentation: Simplon programmer's guide.

Accessing the camera (feature control and image acquisition) using 3rd party software packages compatible with GenICam GenTL and GigE Vision standards is documented in chapter Chapter 8, *Connecting with 3rd party software packages* [p. 215].

6. Camera features and their control

6.1. Common principles

The camera features are controlled through GenICam based feature tree. For clarity, we will present them below in sections corresponding to individual categories in the feature tree. For individual features, we'll provide a sample code showing how to access given feature through the Simplon API and “pseudo-code” demonstrating how to use the feature in a generic GenICam based application. Naming and grouping the features strictly follows the GenICam SFNC standard. The Simplon API samples are provided in plain C, other variants can be easily generated by the Simplon Source Code Generator (refer to Simplon programmer's guide).

The code examples are marked with corresponding icons:

Introduces sample code demonstrating usage of the discussed features through the Simplon API interface. The code snippets are using directly the “native” Simplon API functions and constants. They are presented in the plain C variant, but users programming in other languages should find all needed info in them easily. Whenever a “shortcut” function exists for given functionality in Simplon API, it is presented.

Pseudocode demonstrating usage of the discussed features through a generic GenICam interface. These code snippets are intended for users accessing the cameras through other software interface than Simplon API. It uses the actual GenICam names of the individual features, rather than Simplon API constants. The “pseudo function calls” need to be replaced with actual function names used in the actually used software package.

The sections below document all the features available within the camera family. Some of the features, are, however, only available on certain camera models. Some features might also not be available on older firmware versions. Simplon API provides means to check if a given feature is supported by the connected camera.

Simplon API can add additional “capabilities” to the connected cameras. For example, it can supplement image processing capabilities (such as LUTs or color transformation) by means of software if given camera does not implement them. Simplon API also provides additional extras on top of the processing primitives offered by a camera — examples are white balance coefficients calculations, brightness/contrast adjustments through LUT etc. Details are discussed in Simplon programmer's guide.

The camera feature tree control is based on principles introduced in the GenICam standard. It might be useful to get familiar with these principles in corresponding section in Simplon getting started maual.

The camera feature model follows the GenICam SFNC (Standard Features Naming Convention) standard as closely as possible. Only for functionality that cannot be expressed by means of SFNC, then custom feature names are introduced — these have always the “Lv” prefix.

Individual features might be referred through three types of identifiers in the manual (overview of all the features is given in Section 6.16, “Feature summary” [p. 167]):

- GUI name (display name), eg. *Exposure time* — this is the human readable name of the feature, used to display the feature in a graphical user interface, such as in Simplon Explorer. When referring to the features in a free text, we usually use the GUI name, unless using the GenICam name is needed for clarity. The name is rendered in a cursive font.
- GenICam name, eg. `ExposureTime` — the actual name of the feature used to identify it in GenICam infrastructure. This name is important for users connecting to the cameras from 3rd

party software packages, other than Simplon API. We use it in all GenICam pseudo-code examples. In free text they are rendered in a monospace font.
- Simplon API constant, eg. `LvDevice_ExposureTime` — the identifier used to refer to the feature in Simplon API. Used in all Simplon API code examples. In free text they are rendered in a monospace font.

6.2. Camera classes

The individual Leutron Vision camera families contain several “classes” of camera models, providing different level of functionality. The individual classes are available for every supported sensor type.

6.2.1. Basic CheckSight/CheckSight-II models

Most of the features and interfaces are available for all the camera classes. The “basic” CheckSight/CheckSight-II models are thus full featured industrial smart cameras without any limitations. OEM designed camera models may extend the camera functionality by specific additional options.

6.2.2. OEM features

Additional features and options are available for OEM customers, new features might be added on request. Please contact us to discuss your particular needs.

6.3. Information about the camera

Most of the features in this category provide read-only information about the camera. Only few of them can be writable as well.

Figure 6.1. Example GUI showing the device information features

6.3.1. Device identity and version

The basic information about the camera includes:

Vendor Name — Name of the camera's vendor (manufacturer). For all Leutron Vision camera types, this feature reads `Leutron Vision`.

Model Name — Name of the camera model. The name includes information about the camera family, sensor type, interface and possibly other variations. Refer to Section 2.2, “CheckSight model naming convention” [p. 19].

Manufacturer Info — Manufacturer information about the camera. This field is usually empty.

Device Firmware Version — Version of the camera firmware as a string. The version is encoded as `M.mm.sss`, where `M` stands for major, `mm` for minor and `sss` for subminor version. For example `2.00.005`.

Recovery Firmware Version — Firmware version into which the camera would be recovered. Besides the actual firmware, the camera carries also an older firmware version into which it can be recovered in case of serious problems.

Device Hardware Version — String holding various information about the camera hardware version. On some cameras it might be empty. The contents is intended for our support team.

Device ID (serial number) — String representation of camera's unique serial number.

Beware, the Simplon API identifier for this feature is `LvDevice_DeviceSerialNumber` to distinguish it from the GenICam GenTL "device ID" used for device enumeration.

Device User ID (nickname) — Writable string, which can be used to assign a unique camera "nickname" by the user. The nickname can be up to 15 characters long and it is one of the options how to find/identify a camera in Simplon API.

Device Scan Type — Scanning type of a camera. This SFNC (Standard Features Naming Convention) feature is intended for generic applications — it returns *Area Scan* for all Leutron Vision cameras.

Device Type — String representation of camera type/class. The type associates the camera with a given level of functionality (independent on the mounted sensor). Example values are `GigE`, `GigE-RTF`, or `Smart`, see Section 6.2, "Camera classes" [p. 80].

Simplon

```
LvHDevice hDev; // open the device to get the handle
// read basic information
char StringVal[1024];
LvGetString (hDev, LvDevice_DeviceVendorName, StringVal, 1024);
LvGetString (hDev, LvDevice_DeviceModelName, StringVal, 1024);
LvGetString (hDev, LvDevice_DeviceManufacturerInfo, StringVal, 1024);
LvGetString (hDev, LvDevice_DeviceFirmwareVersion, StringVal, 1024);
LvGetString (hDev, LvDevice_LvRecoveryFirmwareVersion, StringVal, 1024);
LvGetString (hDev, LvDevice_DeviceVersion, StringVal, 1024);
LvGetString (hDev, LvDevice_DeviceSerialNumber, StringVal, 1024);
LvGetString (hDev, LvDevice_DeviceUserID, StringVal, 1024);
LvGetString (hDev, LvDevice_LvDeviceType, StringVal, 1024);
// set new nickname (user ID)
LvSetString (hDev, LvDevice_DeviceUserID, "FoobarCamera");
// check the scanning type (always LvDeviceScanType_Areascan)
int32_t EnumVal;
LvGetEnum (hDev, LvDevice_DeviceScanType, &EnumVal);
```

GEN<i>CAM

```
// read basic information
read_string (DeviceVendorName)
```

```
read_string (DeviceModelName)
read_string (DeviceManufacturerInfo)
read_string (DeviceFirmwareVersion)
read_string (LvRecoveryFirmwareVersion)
read_string (DeviceVersion)
read_string (DeviceID)
read_string (DeviceUserID)
read_string (LvDeviceType)
// set new nickname (user ID)
write_string (DeviceUserID, "FoobarCamera")
// check the scanning type (always AreaScan)
read_enum (DeviceScanType)
```

6.3.2. Runtime information

Apart of the basic information, the camera also provides various “runtime” status information:

Temperature The camera self-diagnostics includes temperature measurement at various locations. Currently reported measurements include temperature at the sensor board and at the main board. Note that before reading the actual value, it is necessary to first adjust the *Temperature Selector* to correct measurement point.

Up Time Current up-time of the device (time period elapsed since camera boot), as integer value in milliseconds.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// read sensor temperature
LvSetEnum (hDev, LvDevice_DeviceTemperatureSelector,
                 LvDeviceTemperatureSelector_Sensor);
double FloatVal;
LvGetFloat (hDev, LvDevice_DeviceTemperature, &FloatVal);
// read main board temperature
LvSetEnum (hDev, LvDevice_DeviceTemperatureSelector,
                 LvDeviceTemperatureSelector_Mainboard);
LvGetFloat (hDev, LvDevice_DeviceTemperature, &FloatVal);
// read device uptime
int32_t IntVal;
LvGetInt32 (hDev, LvDevice_LvDeviceUpTime, &IntVal);
```

GEN<i>CAM

```
// read sensor temperature
write_enum (DeviceTemperatureSelector, Sensor)
read_float (DeviceTemperature)
// read main board temperature
write_enum (DeviceTemperatureSelector, Mainboard)
read_float (DeviceTemperature)
// read device uptime
read_int (LvDeviceUpTime)
```

6.3.3. Resetting the camera

It is possible to reset (reboot) the camera using the *Reset Device* command. It is important to remember, that after issuing the command, the connection with the camera is lost immediately and

it needs to be re-opened after it is booted again. Any temporary camera settings will be lost, the camera will start in the default settings (note that the default startup settings of the camera are user-configurable, see Section 6.13, “User sets (saving/loading camera configuration)” [p. 157]). The camera also performs complete initialization procedure, such as eg. IP configuration (when using DHCP, it can boot with different IP address).

Simplon

```
LvHDevice hDev; // open the device to get the handle
// reset the camera
LvCmdExecute (hDev, LvDevice_DeviceReset);
```

GEN<i>CAM

```
// reset the camera
exec_cmd (DeviceReset)
```

6.3.4. Pixel clock (sensor digitization clock)

Pixel clock feature is a bit special since while it's typically a read-only informative feature, on certain camera models and firmware versions, it might be configurable (section Section 6.6, “Acquisition and trigger control” [p. 101] discusses other options to configure acquisition frame rate). To access the pixel clock frequency through the *Clock Frequency* floating point feature, it's first necessary to set *Clock Selector* to *Sensor Digitization* to differentiate from other clocks a camera might be reporting.

Simplon

```
LvHDevice hDev; // open the device to get the handle
//read camera sensor digitization clock in Hz
LvSetEnum (hDev, LvDevice_DeviceClockSelector,
                 LvDevicClockSelector_SensorDigitization);
LvGetFloat (hDev, LvDevice_DeviceClockFrequency, &FloatVal);
// set pixel clock if the camera supports it (float in Hz)
LvSetEnum (hDev, LvDevice_DeviceClockSelector,
                 LvDevicClockSelector_SensorDigitization);
LvSetFloat (hDev, LvDevice_DeviceClockFrequency, 28.6*1000000);
```

GEN<i>CAM

```
// read camera sensor digitization clock in Hz
write_enum (DeviceClockSelector, SensorDigitization)
read_float (DeviceClockFrequency)
// set pixel clock if the camera supports it (float in Hz)
write_enum (DeviceClockSelector, SensorDigitization)
write_float (DeviceClockFrequency, 28.6*1000000)
```

6.3.5. Register streaming support

The camera supports so called “register streaming”. This GenICam term refers to safe storing and re-loading entire camera configuration. Safe means that it restores complete “state” of the camera, without need to know about complex feature dependencies. The same settings can be applied to multiple cameras of the same type and version. To work reliably, the operation must exactly follow rules standardized by the GenICam standard.

When using the Simplon API, the camera settings might be stored and re-loaded using functions `LvCameraSaveSettings()` and `LvCameraLoadSettings()`, so the user need's not care about the internal meachanism.

Other GenICam based software packages will usually also provide convenience functions to handle camera settings. The camera features controlling the standardized register streaming operation are: `DeviceRegistersStreamingStart`, `DeviceRegistersStreamingEnd`, `DeviceRegistersCheck` and `DeviceRegistersValid`.

Note that the camera itself is also able to remember its current state and restore it later upon user request or during boot, see Section 6.13, “User sets (saving/loading camera configuration)” [p. 157].

6.4. Transport layer control

The most important feature in this category is *Payload Size*, which is used to determine the size of the image data during buffer allocation. Besides that, the category contains lot of “transport layer” (GigE Vision) related features, most of which are handled internally by Simplon API or other imaging library, or can be configured comfortably through Simplon GigE Vision Configuration Tool.

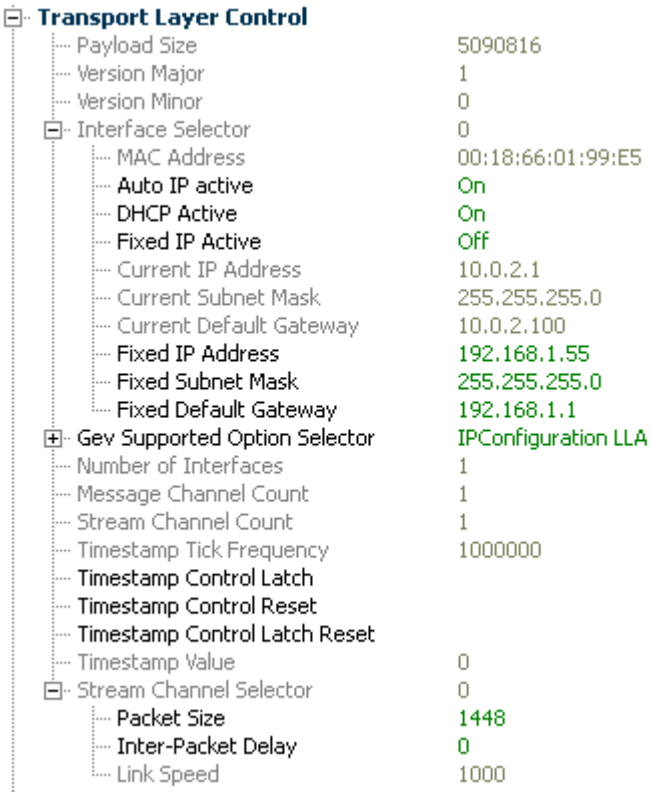


Figure 6.2. Example GUI showing the transport layer control features

6.4.1. Acquired image data size

When preparing for acquisition, the application needs to know the required size of the buffers to be allocated for receiving the acquired data. The size depends on various options, such as image size and pixel format (Section 6.5, “Image format” [p. 85]) or presence of chunk data in the image (Section 6.11, “Chunk data” [p. 146]).

The camera reports the image data size corresponding with current configuration in the *Payload Size* feature, which provides the data size in bytes.

The camera allows to switch to a mode when some of the parameters (eg. image size) affecting the payload size can change freely during acquisition. In such mode the *Payload Size* feature always reports the maximal possible size, so that this size can always be safely used for buffer allocation.

It is important, that the actual value of the *Payload Size* feature does not change during active acquisition — otherwise the application is in danger that its buffers will not be large enough for the received data. GenICam provides a standard way to "lock" the essential "transport layer" features during acquisition through feature called `TLParamsLocked`. Note that this feature is set transparently during acquisition by Simplon API — and in all reasonable GenICam based software packages, so the user does not need to care about it.

Simplon API offers additional image processing functions that can also alter the image data size — and thus the required size of the target buffer. This topic is, however, behind the scope of this manual, it is discussed in Simplon programmer's guide.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// read the "payload size"
// (buffer size required to hold acquired image data)
int32_t IntVal;
LvGetInt32 (hDev, LvDevice_PayloadSize, &IntVal);
// now allocate the buffers and use them for the acquisition
```

GENICAM

```
// read the "payload size"
// (buffer size required to hold acquired image data)
read_int (PayloadSize)
// now allocate the buffers and use them for the acquisition
```

6.5. Image format

The image format features allows to configure image size and all related features, as well as the pixel format.

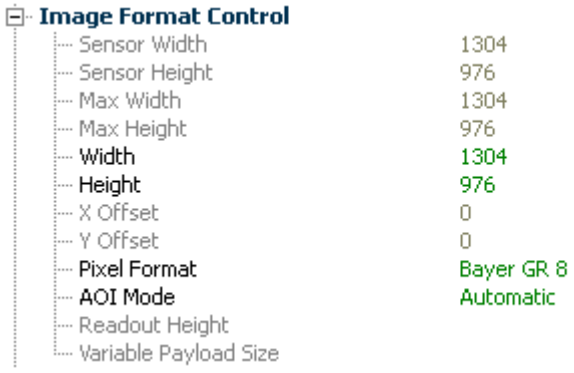


Figure 6.3. Example GUI showing the image format features

6.5.1. Pixel format

The *Pixel Format* feature controls the nature and layout of pixel data in the image buffer. The set of available pixel formats is strongly dependent on the camera model, in particular on the mounted

sensor and also on the image preprocessing functions implemented by given camera, such as Bayer decoding. Generic applications not designed for a single particular camera type should take this into account and query first the available pixel formats.

Simplon API recognizes all the pixel format types and offers functions to process, convert or display them.

The pixel formats fall into following main groups:

- **Monochrome.** The camera provides true monochrome image (luminance only). Depending on the pixel depth, ie. number of bits representing each pixel, the monochrome formats can be `Mono8`, `Mono10` or `Mono12`.

  Bayer encoded color. The camera provides color image as raw Bayer mosaic encoded data. Similarly as for monochrome formats, it the Bayer formats also exist in various bit depths. Furthermore, individual image sensors might layout the color components different way within the image — depending on the two pixels in top-left image corner, the formats are named:
  - RG — the top left pixel is red
  - BG — the top left pixel is blue
  - GR — the top left pixel is green, followed by a red one
  - GB — the top left pixel is green, followed by a blue one

  The most usual Bayer encoded formats are `BayerRG8`, `BayerRG12` and `BayerRG10`. Simplon API provides functions for converting Bayer encoded data to true RGB image.

  True color. The camera provides “true” color image in 32-bit BGR format (blue component first), 8-bits per color channel. Name of the format is `BGRA8Packed`.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// set pixel format "Mono8", if available
if (LvIsAvailableEnumEntry(hDev, LvDevice_PixelFormat,
                                 LvPixelFormat_Mono))
{
    LvSetEnum (hDev, LvDevice_PixelFormat, LvPixelFormat_Mono8);
}
```

GEN<i>CAM

```
// set pixel format "Mono8"
write_enum (PixelFormat, Mono8)
```

6.5.2. Binning and decimation

Binning and decimation are two different sub-sampling operations performed directly at the camera sensor, before image readout. They reduce the horizontal and/or vertical resolution by given factor, thus yielding higher frame rate while preserving the field of view. Compare that with area of interest (AOI) settings, which reduce field of view.

Different factors might be used for horizontal and vertical sub-sampling (yielding a distorted image). Note that the availability and range of possible sub-sampling factors varies among individual camera models. Some color sensors are not capable of binning/decimation even if their monochrome counterparts are.

Let's have a look, how binning and decimation is implemented:

Binning — Every N neighbouring pixels (N being the binning factor) are merged together to produce a single output pixel. Depending on the binning implementation on each

respective sensor, the pixels might be merged by summing or averaging pixel values. Binning is applied on the analog pixel values, before digitization. It thus improves low-light camera performance, in particular the signal to noise (S/N) ratio. It is controlled through features *Horizontal Binning* and *Vertical Binning*.

Decimation — Simple pixel skipping, only every Nth (N being the decimation factor) pixel is output from the sensor, the others are ignored. Decimation does not affect lightness or S/N ratio. It is controlled through features *Horizontal Decimation* and *Vertical Decimation*.

The following two figures illustrate the differences between binning and decimation, first using a simple schematics, second on a "real life" image. Note that the actual binning result (in particular the resulting image lightness) might differ depending on the binning implementation on different camera sensors.

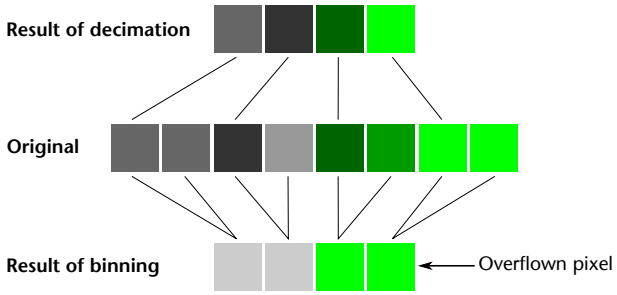


Figure 6.4. Binning and decimation explained (assumes binning by pixel summing)

Figure 6.5. Binning and decimation example

Simplon

```
LvHDevice hDev; // open the device to get the handle
// configure 2x2 binning
LvSetInt32 (hDev, LvDevice_BinningHorizontal, 2);
LvSetInt32 (hDev, LvDevice_BinningVertical, 2);
// configure 2x2 decimation
LvSetInt32 (hDev, LvDevice_DecimationHorizontal, 2);
LvSetInt32 (hDev, LvDevice_DecimationVertical, 2);
```

GEN<i>CAM

```
// configure 2x2 binning
write_int (BinningHorizontal, 2)
write_int (BinningVertical, 2)
// configure 2x2 decimation
write_int (DecimationHorizontal, 2)
write_int (DecimationVertical, 2)
```

Since the binning/decimation might increase the maximal acquisition frame rate, they might have also an impact on the exposure time limits. In free running mode, the maximal exposure time depends on the current frame rate — it cannot be increased beyond a point, when longer exposures would slow down the acquisition. In conclusion, adjusting binning/decimation might lead to lowering the exposure time range and the exposure time would be automatically adjusted to fit in the new range. In triggered mode, the exposure time limits are not affected.

6.5.2.1. Binning/decimation on color cameras

The color (Bayer encoded) cameras supporting binning and/or decimation implement it intelligently, merging/skipping pixels of corresponding colors in the Bayer mosaic filter. The output is thus always a correct Bayer pattern compatible with the natural pattern of the sensor. The color representation is not destroyed.

6.5.3. Image size, area of interest (AOI)

Another option to reduce image resolution is area of interest (AOI) feature. Area of interest is a configurable rectangular area within the image sensor, that is read out and transferred to the application buffer. Pixels outside this defined area are ignored. Contrary to binning/decimation, AOI reduces the field of view. The effect on frame rate depends on the camera model, size and position of the AOI rectangle within the sensor.

The AOI functionality is available on all camera models.

The area of interest is defined by four basic parameters, specifying the size and offset within the “parent” area (which is, in the most simple case, full sensor size):

Width/Height	Dimensions of the area of interest, in pixels.
X Offset/Y Offset	Offset of the area of interest, relative to the top-left corner of the parent area (sensor), in pixels.

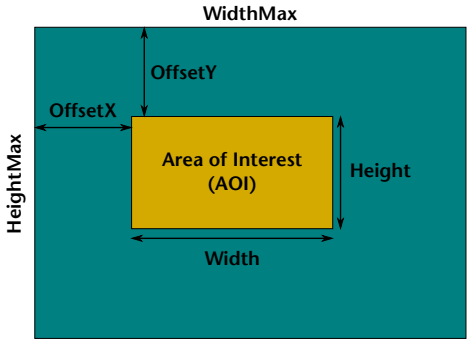


Figure 6.6. AOI parameters

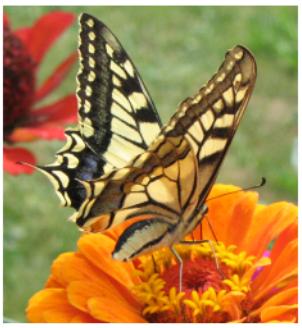


Figure 6.7. AOI example

In the default mode, only the AOI size is configurable, the x/y offsets (position within the sensor area) are selected automatically by the camera to optimize performance. This typically means that the area of interest is centered within the sensor. On some cameras only *Height* is configurable.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// decrease AOI size to 300x200 pixels
LvSetInt32 (hDev, LvDevice_Width, 300);
LvSetInt32 (hDev, LvDevice_Height, 200);
// read current maximum AOI size
// (depends on binning/decimation, see below)
```

```
int32_t IntVal;
LvGetInt32 (hDev, LvDevice_WidthMax, &IntVal);
LvGetInt32 (hDev, LvDevice_HeightMax, &IntVal);
// check entire sensor size (in pixels)
LvGetInt32 (hDev, LvDevice_SensorWidth, &IntVal);
LvGetInt32 (hDev, LvDevice_SensorHeight, &IntVal);
```

GEN<i>CAM

```
// decrease AOI size to 300x200 pixels
write_int (Width, 300)
write_int (Height, 200)
// read current maximum AOI size
// (depends on binning/decimation, see below)
read_int (WidthMax)
read_int (HeightMax)
// check entire sensor size (in pixels)
read_int (SensorWidth)
read_int (SensorHeight)
```

More elaborate options to configure area of interest will be shortly discussed below.

Since the AOI settings might increase the maximal acquisition frame rate, they might have also an impact on the exposure time limits. In free running mode, the maximal exposure time depends on the current frame rate — it cannot be increased beyond a point, when longer exposures would slow down the acquisition. In conclusion, adjusting AOI might lead to lowering the exposure time range and the exposure time would be automatically adjusted to fit in the new range. In triggered mode, the exposure time limits are not affected.

When decreasing the AOI to very small size, the acquisition frame rate can increase significantly, having unwanted impact on system performance due to increased overhead. The actual behavior is strongly dependent on the used sensor, system configuration and other conditions and cannot be exactly quantified. The developer should consider possible impact of using extremely small AOI size on the system performance and reliability.

6.5.3.1. AOI limits

Two pairs of helper read-only features provide information related to maximal size of the area of interest under given conditions:

Sensor Width/*Sensor Height*
: Effective resolution of the image sensor, specifying the number of useful pixels that can be read out of the sensor. These values are fixed for a given camera model. In the default mode (no binning, decimation, or area of interest), the image dimensions equal to *Sensor Width* and *Sensor Height*.

Max Width/*Max Height*
: These values specify the maximum possible size of the area of interest in current configuration. This maximum size can be affected by features such as binning, which reduce the effective sensor size. When no binning/decimation is active, *Max Width*×*Max Height* are equal to *Sensor Width*×*Sensor Height*.

6.5.3.2. AOI and binning/decimation

When binning or decimation is active, the area of interest is applied to the virtual sensor area resulting from the binning/decimation operation. In this case, the *Max Width* and *Max Height* features

report the size resulting from binning/decimation, ie. sensor size reduced by given factor(s). Take this in account when using the two features together.

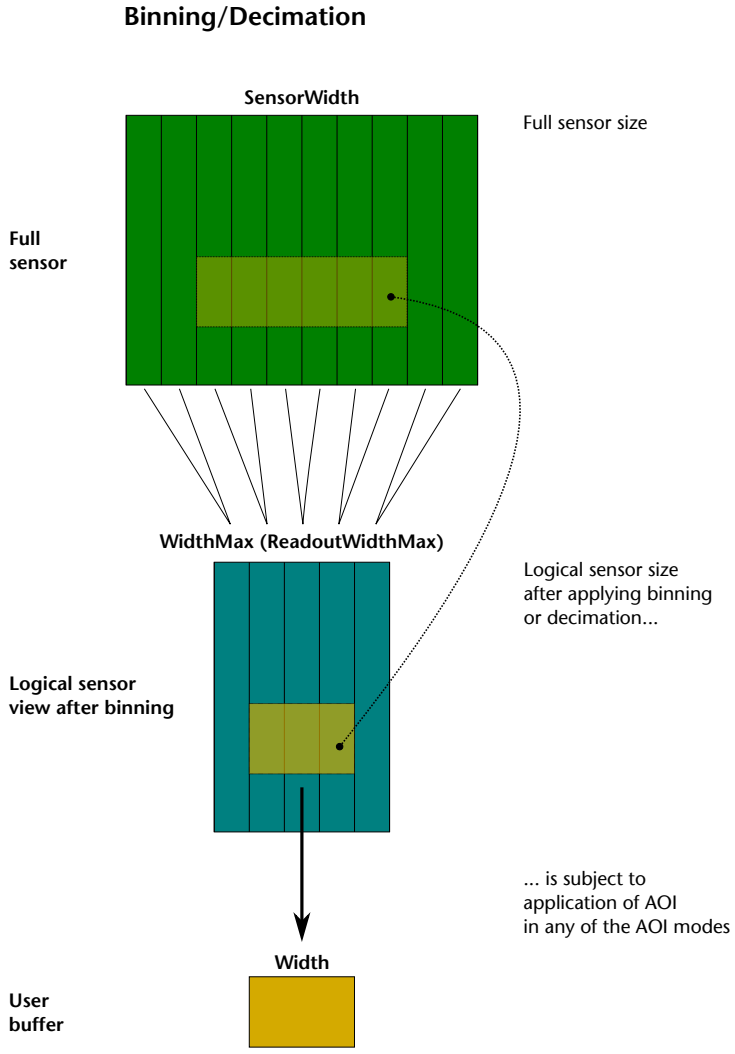


Figure 6.8. AOI and binning

6.5.3.3. AOI on color cameras

On the color (Bayer encoded) cameras, the AOI settings, size an offset, can only be adjusted with increment of 2 to keep the same Bayer mosaic format in the buffer, regardless the actual AOI configuration.

6.5.4. Advanced AOI settings

While the default “automatic” mode described above is sufficient in most cases, some applications might require greater control over the AOI settings.

6.5.4.1. Two-step AOI implementation

In general, there are two points, where area of interest can be applied:

Sensor — The camera might configure the sensor to read out only the specified area of interest. This attitude has the advantage, that the AOI has direct impact on the frame rate (the smaller AOI, the faster readout and frame rate).

On the other hand, this implementation is strongly sensor specific. Some sensors do not allow to reduce AOI in both directions, others impose additional constraints, such as that the AOI has to be centered within the sensor. The functionality would thus be quite limited on some camera models. It is not allowed to reconfigure sensor-side AOI during active acquisition.

Camera frame buffer

The camera could alternatively read out the entire image from the sensor into its frame buffer and apply the area of interest (clip the image) before transferring it out to the host memory (application buffer). In this case, the AOI settings have no impact on the sensor readout time and frame rate. Only the required bandwidth for image transfer decreases — that means lower network load for PicSight-GigE, lower PCI/DMA load for CheckSight.

The advantage of this attitude is that the AOI can be configured freely, without any constraints. The AOI parameters can even be, if desired, changed on-the-fly, during active acquisition. The programmer just needs to understand all the consequences of variable image size during acquisition — this topic is discussed in Section 6.5.5, “Fixed vs. variable payload size” [p. 99].

Leutron Vision cameras allow to combine both attitudes flexibly through various “AOI modes”.

6.5.4.2. AOI modes

The AOI modes are configured through the enumerated feature *AOI Mode*. Depending on the selected AOI modes, the camera decides, how to interpret the AOI parameters, or even enables additional parameters.

The available modes are:

- *Automatic*
- *Clip on Transfer*
- *Manual*

Automatic AOI mode

This is the default mode and was described above (Section 6.5.3, “Image size, area of interest (AOI)” [p. 88]).

- Only the AOI size (*Width*/*Height*) is configurable. Its position (*X Offset*/*Y Offset*) is chosen by the camera — usually centered within the sensor, unless the sensor disallows that.
- The camera automatically “distributes” the AOI implementation among the two options, sensor-side and in-memory-clip. It tries to apply as much as possible directly on the sensor to increase the frame rate as much as possible. If sensor AOI capabilities are limited, it additionally clips the image in-memory before the transfer, to reach the desired AOI settings.
- The AOI parameters cannot be changed during active acquisition.

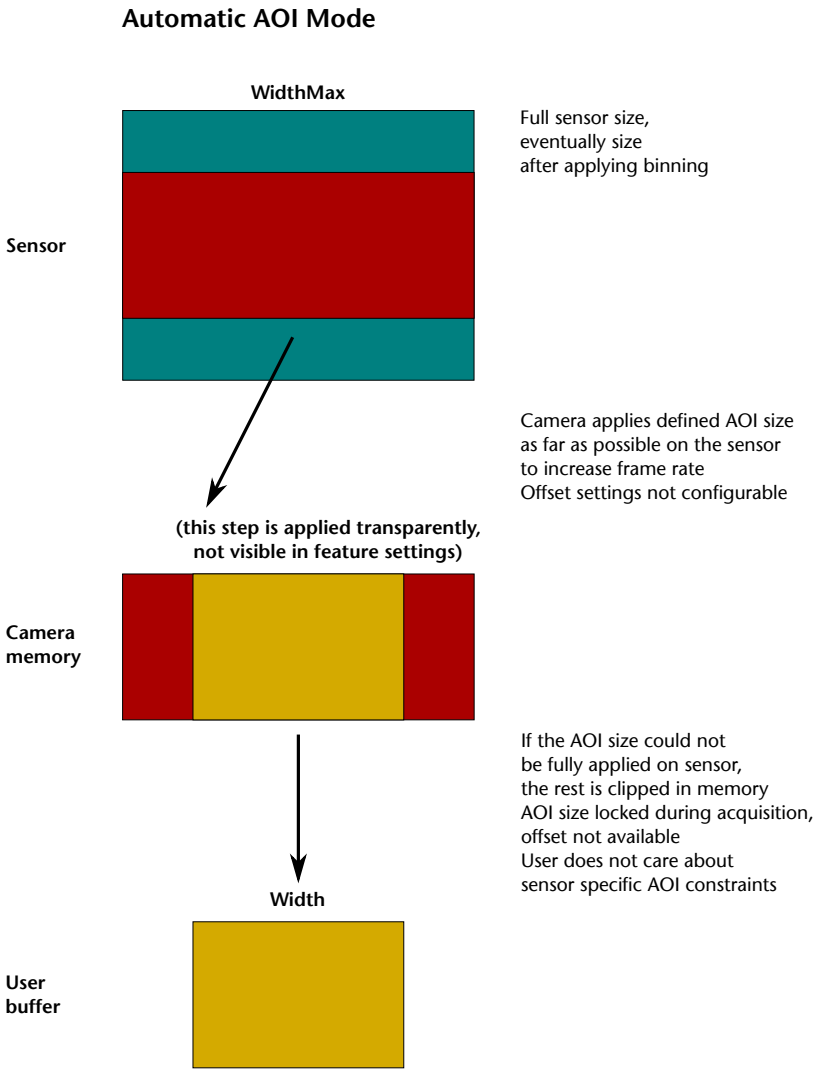


Figure 6.9. Automatic AOI mode

Simplon

```
LvHDevice hDev; // open the device to get the handle
// select automatic AOI mode
LvSetEnum (hDev, LvDevice_LvAOIMode, LvAOIMode_Automatic);
// adjust AOI size to 300x200 pixels
LvSetInt32 (hDev, LvDevice_Width, 300);
LvSetInt32 (hDev, LvDevice_Height, 200);
```

GEN<i>CAM

```
// select automatic AOI mode
write_enum (LvAOIMode, Automatic)
// adjust AOI size to 300x200 pixels
write_int (Width, 300)
write_int (Height, 200)
```

Clip-on-transfer AOI mode

This mode allows free control over AOI parameters, optionally even at runtime. It might not be implemented on all camera models since the same functionality can be achieved using the "manual" mode.

- The AOI is fully applied in camera memory — the image is "clipped" before transferring out of the camera.
- Since sensor settings are not involved, the AOI has no positive impact on the frame rate.
- All four AOI parameters, *Width*/*Height* and *X Offset*/*Y Offset*, are freely configurable. No sensor specific constraints apply.
- Offsets are configurable during active acquisition.
- Size can be configurable during acquisition, see Section 6.5.5, "Fixed vs. variable payload size" [p. 99].

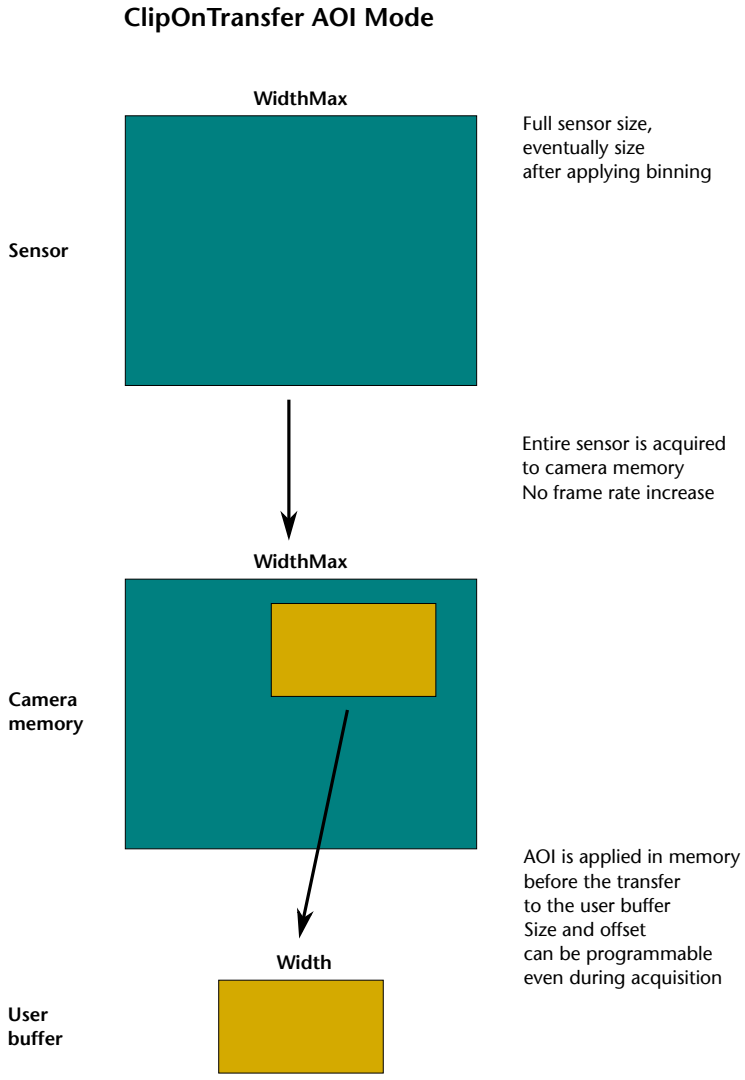


Figure 6.10. Clip-on-transfer AOI mode

```
LvHDevice hDev; // open the device to get the handle
// select clip-on-transfer AOI mode
LvSetEnum (hDev, LvDevice_LvAOIMode, LvAOIMode_ClipOnTransfer);
```

```
// adjust AOI size to 300x200 pixels
// with offset 100 pixels from top and left image sides
// (note that the order of setting the size and offset values
// might matter as described later)
LvSetInt32 (hDev, LvDevice_Width, 300);
LvSetInt32 (hDev, LvDevice_Height, 200);
LvSetInt32 (hDev, LvDevice_OffsetX, 100);
LvSetInt32 (hDev, LvDevice_OffsetY, 100);
```

GEN<i>CAM

```
// select clip-on-transfer AOI mode
write_enum (LvAOIMode, ClipOnTransfer)
// adjust AOI size to 300x200 pixels
// with offset 100 pixels from top and left image sides
// (note that the order of setting the size and offset values
// might matter as described later)
write_int (Width, 300)
write_int (Height, 200)
write_int (OffsetX, 100)
write_int (OffsetY, 100)
```

Manual AOI mode

The most flexible mode, allowing elaborate options for AOI configuration.

- The AOI can be configured in two separate steps: sensor-readout and in-memory-clip AOI. The first one defines the region to be read out from a sensor into the camera memory. The second AOI is applied to this region directly in camera memory (clipped before the transfer).
- The sensor-readout AOI is tightly coupled with capabilities of each respective sensor. All sensor specific constraints apply — examples: only height is configurable, the AOI has to be centered, etc.
- Four additional parameters are introduced in the manual AOI mode to define the sensor-readout AOI part: *Readout Width*, *Readout Height*, *Readout XOffset* and *Readout Y Offset*. If no binning and decimation is applied, the size limits are equal to the sensor size (*Sensor Width*/*Sensor Height*). When using binning or decimation, the size limits are defined by the result of the binning/decimation operation (sensor size divided by the binning/decimation factor).
- The readout AOI parameters are not configurable during active acquisition.
- The second step is performed in the camera's frame buffer memory, after readout, and is equivalent to the “clip-on-transfer” AOI mode. It has all properties of that mode — in particular: the parameters are freely configurable without constraints, offsets are configurable during acquisition, size might be configurable during acquisition (Section 6.5.5, “Fixed vs. variable payload size” [p. 99]).
- The “in-memory-clip” AOI step is configured through the “regular” AOI parameters, ie. *Width*/*Height* and *Y Offset*/*Y Offset*. The AOI size limits are reported through *Max Width* and *Max Height*, which are defined by the readout AOI size.

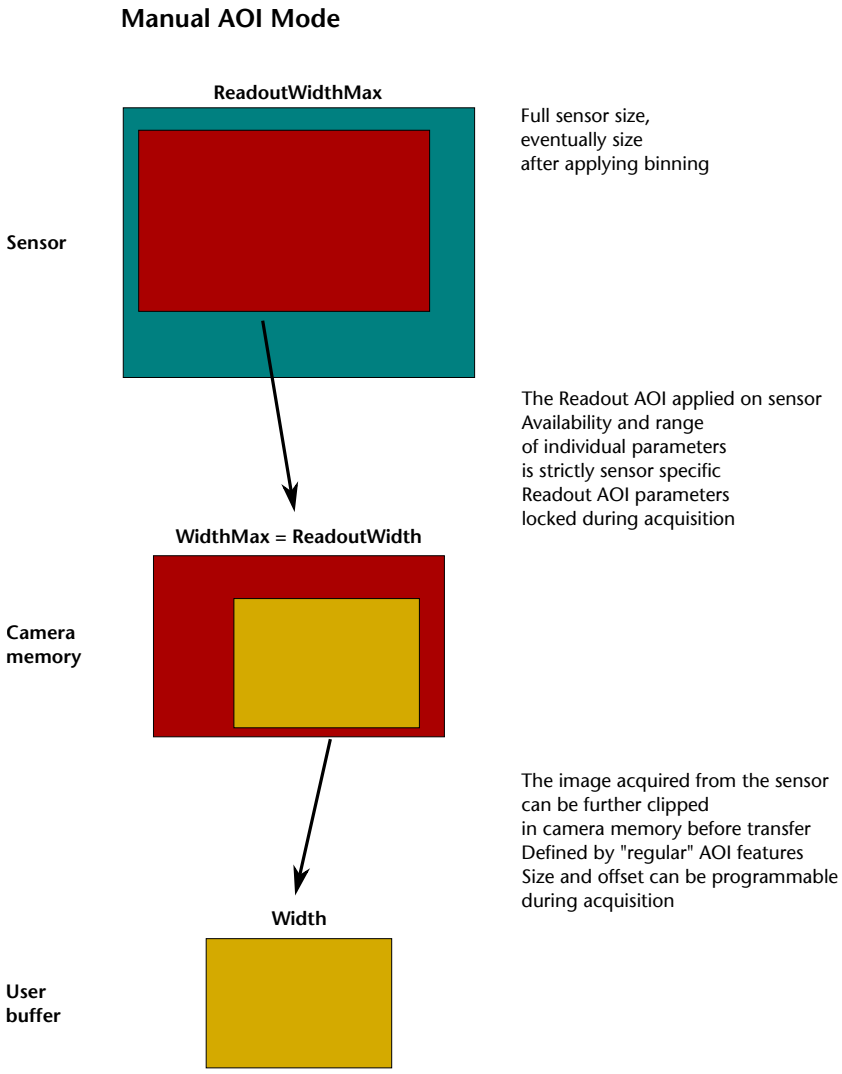


Figure 6.11. Manual AOI mode

```
LvHDevice hDev; // open the device to get the handle
// select manual AOI mode
LvSetEnum (hDev, LvDevice_LvAOIMode, LvAOIMode_Manual);
// adjust the sensor readout AOI parameters
// (note that on most sensors only a subset of the parameters
// would be configurable)
LvSetInt32 (hDev, LvDevice_LvReadoutWidth, 600);
LvSetInt32 (hDev, LvDevice_LvReadoutHeight, 400);
LvSetInt32 (hDev, LvDevice_LvReadoutOffsetX, 200);
LvSetInt32 (hDev, LvDevice_LvReadoutOffsetY, 200);
// now adjust the in-memory-clip AOI parameters
// (note again that the order of setting the size and offset values
// might matter as described later)
LvSetInt32 (hDev, LvDevice_Width, 300);
LvSetInt32 (hDev, LvDevice_Height, 200);
LvSetInt32 (hDev, LvDevice_OffsetX, 100);
LvSetInt32 (hDev, LvDevice_OffsetY, 100);
```

GEN<i>CAM

```
// select manual AOI mode
write_enum (LvAOIMode, Manual)
// adjust the sensor readout AOI parameters
// (note that on most sensors only a subset of the parameters
// would be configurable)
write_int (LvReadoutWidth, 600)
write_int (LvReadoutHeight, 400)
write_int (LvReadoutOffsetX, 200)
write_int (LvReadoutOffsetY, 200)
// now adjust the in-memory-clip AOI parameters
// (note again that the order of setting the size and offset values
// might matter as described later)
write_int (Width, 300)
write_int (Height, 200)
write_int (OffsetX, 100)
write_int (OffsetY, 100)
```

Order of image format settings

When combining multiple image format settings, it is important to adjust them in correct order, since each step defines the limits for the following step, and a virtual intermediate image to which the consecutive step is applied. The correct procedure is:

1. **AOI mode.** The AOI mode defines, which parameters will be configurable and what will be their limits.

2. **Binning/decimation.** The intermediate image resulting from the binning/decimation operation is a basis for the AOI settings.

3. **Readout AOI.** If in manual AOI mode, adjust the readout AOI parameters first. The result of the readout AOI operation defines a basis for the transfer AOI step (in-memory clipping).

4. **Transfer AOI.** This are the “regular” AOI parameters, available in all AOI modes. They define the final image size that will be transferred to the host application buffers.

5. **Other parameters.** The other image format related parameters, such as pixel format are independent from the image size-related settings and can be applied at any point.

Copyright © Leutron Vision

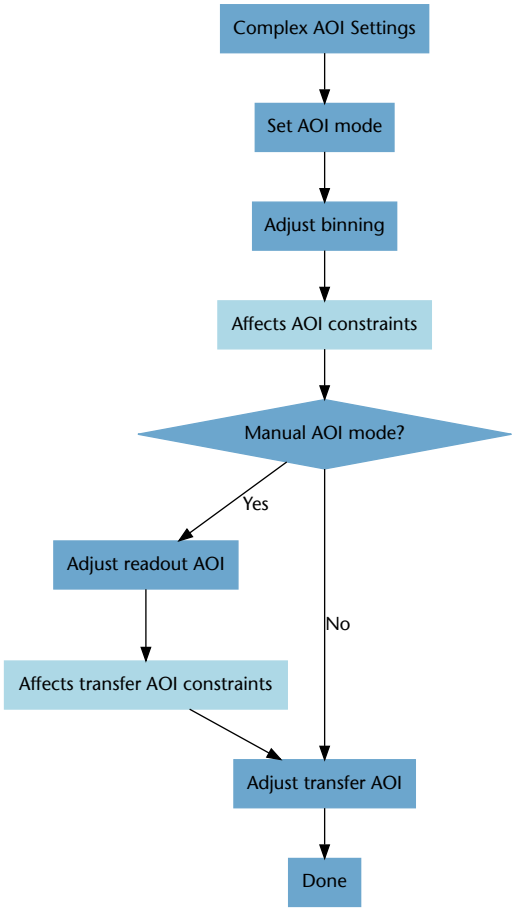


Figure 6.12. Flow chart: image format configuration options

6.5.4.3. AOI parameter dependencies

When working with AOI, it's important to understand the dependencies between the size and offset parameters. The configured area of interest must naturally fall fully into the sensor area. Thus the sum of an offset with its corresponding size parameter must not exceed the current maximum size (which in turn might depend on other features such as binning): `OffsetX+Width<=WidthMax`, `OffsetY+Height<=HeightMax`.

The current *Width* implies the maximum possible *X Offset* and vice versa (similar for vertical settings). For example when AOI width is equal to the sensor size, the horizontal offset must naturally remain `0` — to be able to increase offset, one has to decrease the width first. The limits for the AOI parameters can be expressed in simple equations:

- $\texttt{OffsetX}_{max}$ = WidthMax - Width
- $\texttt{Width}_{max}$ = WidthMax - OffsetX
- $\texttt{OffsetY}_{max}$ = HeightMax - Height
- $\texttt{Height}_{max}$ = HeightMax - OffsetY

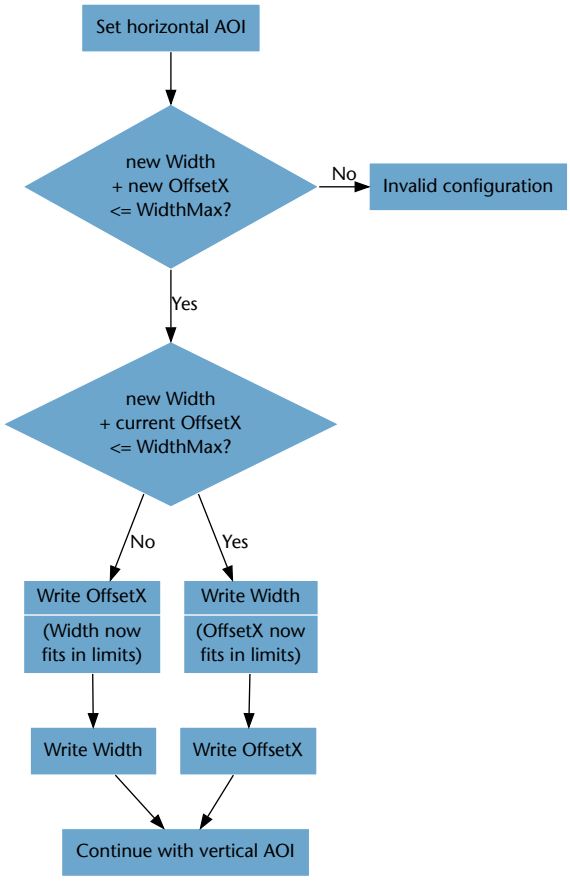


Figure 6.13. Flow chart: AOI parameter dependencies

6.5.5. Fixed vs. variable payload size

The image format features are (together with the chunk data configuration, Section 6.11, “Chunk data” [p. 146]) affecting the size of the data the camera transfers to the host. The application has to allocate buffers large enough to hold the entire data, otherwise the acquisition fails.

The size of the image data (“payload”) the camera is going to transfer is reported in the *Payload Size* feature — the application is expected to use query this reported size when allocating the acquisition buffers.

By default, the payload size is fixed during acquisition, the camera transfers the same amount of data for each acquired image.

If the application desires, however, to change image size during acquisition, special care has to be taken, since the actual amount of data could differ from image to image. Such mode has to be enabled explicitly by setting the *Variable Payload Size* flag. This has two effects:

- The *Width*/*Height* AOI parameters become configurable during acquisition (in manual and clip-on-transfer AOI modes). The image size can change from image to image.
- The *Payload Size* feature does not report the data size corresponding with the *current* image size, but with *maximal* image size. This assures that the reported size is large enough to hold the entire image data, however the image size can change during the acquisition. The buffers allocated using the *Payload Size* will be safe to hold the entire image data during entire acquisition.

The application needs to keep track of the actual image size for every single image, so that it interprets the buffer correctly. It can do so using three mechanisms:

- The camera can report the image size in the chunk data (Section 6.11, “Chunk data” [p. 146]). When connecting the cameras through other software than Simplon API, check if your library supports chunk data decoding.
- The parameters of the image acquired in each respective buffer can be retrieved through the GenTL interface. Simplon API provides direct access to these info fields.
- The GigE Vision compatible cameras (PicSight-GigE and PicSight-Smart) report the image size also in GigE Vision control packets. When connecting the cameras through other software than Simplon API, check if your library provides the information from the control packets and how.

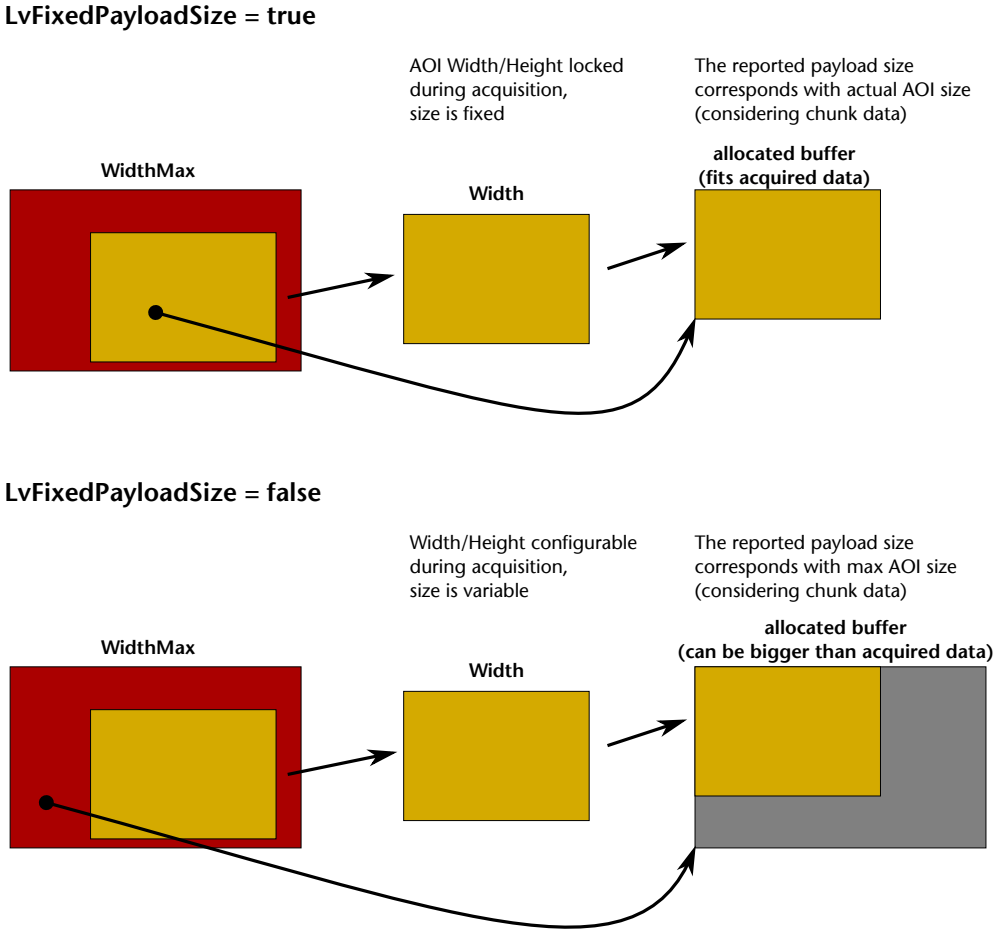

LvFixedPayloadSize = true
AOI Width/Height locked during acquisition, size is fixed
The reported payload size corresponds with actual AOI size (considering chunk data)
WidthMax
Width
allocated buffer (fits acquired data)
LvFixedPayloadSize = false
Width/Height configurable during acquisition, size is variable
The reported payload size corresponds with max AOI size (considering chunk data)
WidthMax
Width
allocated buffer (can be bigger than acquired data)

Figure 6.14. Variable payload size

Simplon

```
LvHDevice hDev; // open the device to get the handle
// use variable payload size
// (the PayloadSize feature reports max possible size)
LvSetBool (hDev, LvDevice_LvVariablePayloadSize, 1);
// use fixed payload size
// (the PayloadSize feature reports actual size)
LvSetBool (hDev, LvDevice_LvVariablePayloadSize, 0);
// ...after finishing the configuration, read payload size feature
// and use it for the buffer allcoation
int32_t IntVal;
LvGetInt32 (hDev, LvDevice_PayloadSize, &IntVal);
```

GEN<i>CAM

```
// use variable payload size
// (the PayloadSize feature reports max possible size)
write_bool (LvVariablePayloadSize, true)
// use fixed payload size
// (the PayloadSize feature reports actual size)
write_bool (LvVariablePayloadSize, false)
// ...after finishing the configuration, read payload size feature
// and use it for the buffer allcoation
read_int (PayloadSize)
```

6.6. Acquisition and trigger control

This section discusses all features related to image acquisition, from essential acquisition control, exposure time, frame rate to trigger modes.

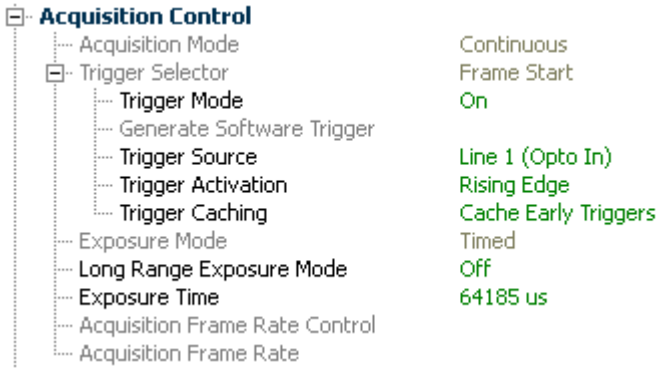


Figure 6.15. Example GUI showing the acquisition and trigger features

6.6.1. Basic acquisition control

The camera delivered off factory always starts in “free running” mode — as soon as the acquisition is started, it acquires images in highest possible frame rate, without need for any triggering.

The basic acquisition control is, in all working modes, done using few basic camera features:

Acquisition Mode — This enumeration is a mandatory feature for GigE Vision cameras — hence it is implemented on all our camera families, even if they currently support only a single acquisition mode, *Continuous*. Setting this feature is thus optional.

Acquisition Start — Command that actually starts the acquisition. In free running mode, the camera starts immediatelly grabbing and streaming images. In triggered mode the images will be acquired upon valid triggers. Simplon API encapsulates this command in `LvDeviceAcquisitionStart()`, together with additional startup actions.

Acquisition Stop — Command stopping the acquisition. In free running mode the acquisition stops after finishing the current frame. In triggered mode no more triggers will be accepted. Simplon API encapsulates this command in `LvDeviceAcquisitionStop()`, together with additional stopping actions.

Acquisition Arm — Optional command that puts the camera to a status ready for acquisition. It performs some initialization and verification routines otherwise handled during acquisition start. It is optional, though, if omitted, if will be ex-

ecuted implicitly during acquisition start. The command is not needed with Simplon API.

`TLParamsLocked` This feature belong to the “transport layer features” category, Section 6.4, “Transport layer control” [p. 84]. Its purpose is to “lock” the features essential for the transport layer (software component transporting the image data to the application buffers). Once set, the crucial acquisition parameters will become locked — most importantly the *Payload Size*, value indicating the required size of the acquisition buffers. Having this feature locked guarantees that the user cannot cause accidental buffer overflows by changing camera configuration.

The GenICam standard dictates that this feature is set throughout the entire acquisition.

Simplon API handles `TLParamsLocked` transparently while starting/stopping the acquisition. When using the camera with another software package, it is important that the feature is set properly before starting the acquisition, otherwise any attempt to start the acquisition will fail (the acquisition start command will not be executable).

In Simplon API, none of these features (with exception of the acquisition mode) are accessed directly, Simplon API does that automatically within the `LvDeviceAcquisitionStart()` and `LvDeviceAcquisitionStop()` functions. The steps needed to initialize a Simplon API application and prepare it for acquisition should be studied in Simplon programmer's guide.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// how to control acquisition:
// optional: set explicitly the acquisition mode
LvSetEnum (hDev, LvDevice_AcquisitionMode, LvAcquisitionMode_Continuous);
// configure essential features such as trigger mode and image format
// perform other basic Simplon initialization, as indicated in its manual
// when finished, read required buffer size and allocate buffers
int32_t IntVal;
LvGetInt32 (hDev, LvDevice_PayloadSize, &IntVal);
// start and stop the acquisition, as desired, Simplon API
// controls the underlying features automatically under the hood
while (NewAcquisitionsDesired)
{
    LvDeviceAcquisitionStart (hDev);
    // ... acquire images as long as desired
    LvDeviceAcquisitionStop (hDev);
}
// if desired, reconfigure the camera and start again
```

In 3rd party GenICam based software package, similar attitude might be expected — ie. the acquisition related features might be hidden and wrapped into dedicated functions. The following pseudocode anyway demonstrates the required procedure to start and stop the acquisition, but remember that a typical GenICam aware library does this completely for you under the hood:

GENICAM

```
// how to control acquisition:
// optional: set explicitly the acquisition mode
write_enum (AcquisitionMode, Continuous)
// configure essential features such as trigger mode and image format
// optional: 'arm' the camera (prepare it for acquisition)
```

```
exec_cmd (AcquisitionArm)
// lock the transport layer parameters, as required by GenICam standard
write_int (TLParamsLocked, 1)
// now it is possible to read the required buffer size, and allocate buffers
// the size will not change as long as TLParamsLocked is set
read_int (PayloadSize)
// start and stop the acquisition, as desired
while (NewAcquisitionsDesired)
{
    exec_cmd (AcquisitionStart)
    // ... acquire images as long as desired
    exec_cmd (AcquisitionStop)
}
// unlock the transport layer parameters
write_int (TLParamsLocked, 0)
// if desired, reconfigure the camera and start again
```

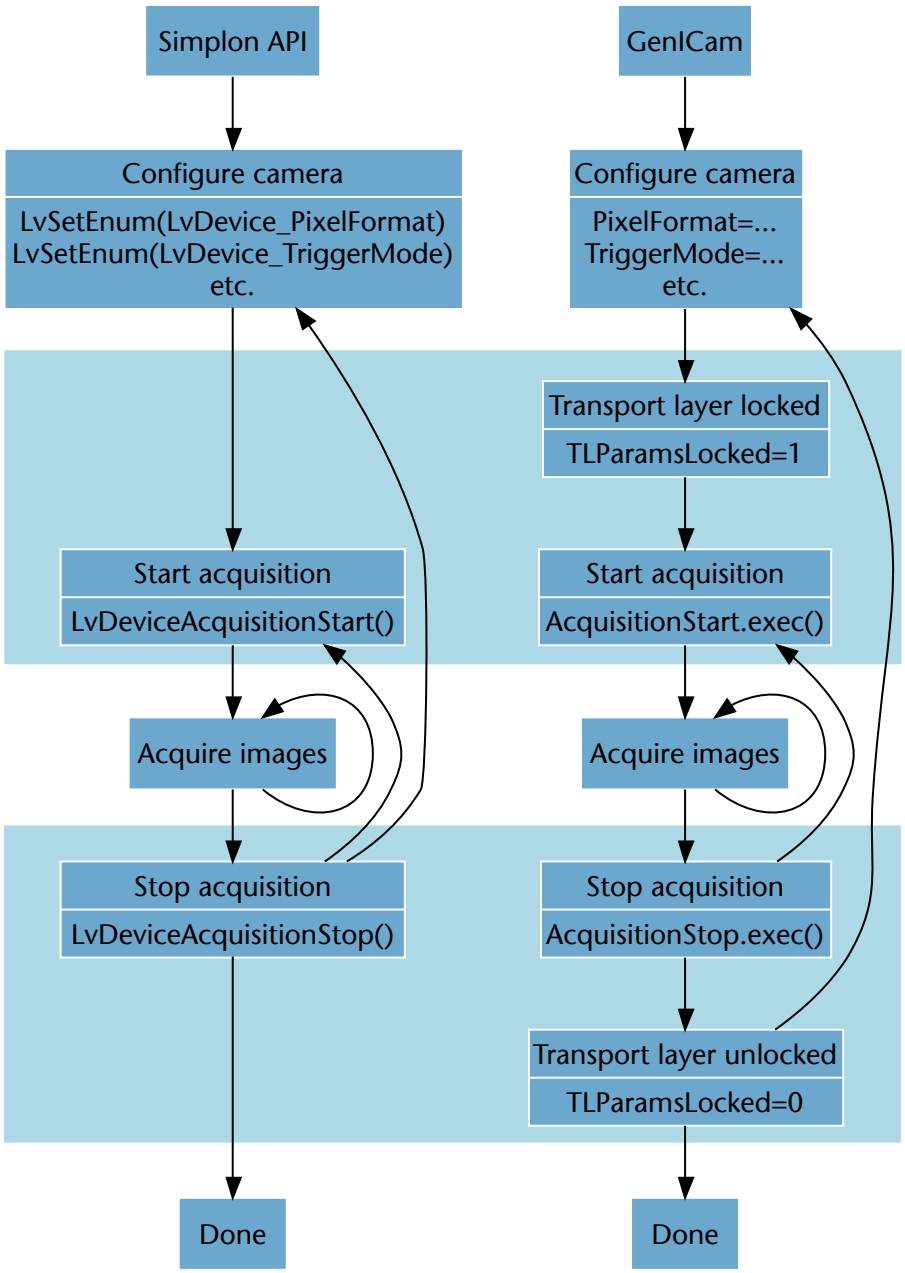


Figure 6.16. Flow chart: acquisition control in Simplon API and in generic GenICam application

6.6.2. Exposure time

The direct setting of exposure time is trivial, through the *Exposure Time* feature, which is a floating point value in µs.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// configure exposure time in microseconds
LvSetFloat (hDev, LvDevice_ExposureTime, 1000.0);
```

GENICAM

```
// configure exposure time in microseconds
write_float (ExposureTime, 1000.0)
```

The actual range of possible exposure times depends on other settings, in particular:

- *Working mode — free running or triggered, long range exposure mode (see below)*
- *Area of interest (AOI) settings*
- *Binning/decimation settings*
- *Frame rate settings*

Note that the camera might be able to adjust the exposure time only in small but discrete steps. It will align the requested value to the closest "step" — to know the exact used value, the application might want to re-read the exposure time after setting it.

On certain camera models (in particular those based on the G43A sensor), the exposure time is applied with a delay of one frame. The one frame delay also applies for sensor R312AB, but only in triggered mode.

Related topic of controlling gain and black level is discussed in separate section, Section 6.9, "Analog controls" [p. 127].

6.6.2.1. Automatic exposure

The automatic exposure function is available only on those camera models, whose sensor supports that function natively. It is useful for applications capturing scenes exposed to significantly varying illumination, when it's necessary to protect the acquired images against under- or over-exposition.

The automatic exposure algorithm observes the luminance of acquired images and attempts to keep it at constant level, by compensating for any changes by adjusting the exposure time for subsequent frames. To prevent flickering, the adjustments are low-pass fltered — it may take several frames until the target luminance is restored again.

To switch the camera to automatic exposure (if supported by the camera), you switch the *Automatic Exposure* enumeration from *Off* to *Continuous*. The exposure time itself will become read-only since that feature is now under control of the camera firmware

Simplon

```
LvHDevice hDev; // open the device to get the handle
// switch to automatic exposure mode
if (LvIsAvailableEnumEntry(hDev, LvDevice_ExposureAuto,
                                 LvExposureAuto_Continuous))
{
    LvSetEnum (hDev, LvDevice_ExposureAuto, LvExposureAuto_Continuous);
}
// switch back to the direct control of exposure time
LvSetEnum (hDev, LvDevice_ExposureAuto, LvExposureAuto_Off);
```

GEN<i>CAM

```
// switch to automatic exposure mode
write_enum (ExposureAuto, Continuous)
// switch back to the direct control of exposure time
write_enum (ExposureAuto, Off)
```

When used together with the automatic gain feature (Section 6.9, "Analog controls" [p. 127]), the auto exposure takes precedence. Automatic gain is inactive as long as the automatic exposure functinality does not reach its limits — only then the automatic gain becomes active as well.

The automatic exposure functionality might work against the manual gain settings. When increasing the gain, the automatic exposure function will decrease the exposure to reach again the target luminance level.

The low pass filter applied in the automatic exposure algorithm can cause, that the automatic exposure value never fully reaches the limits (min/max) of the manual exposure time.

6.6.2.2. Exposure mode

To make the discussion complete, we should also mention the *Exposure Mode* feature. It is an enumeration, which is currently present mainly as information for generic applications relying on the GenICam SFNC (Standard Features Naming Convention) standard — since all Leutron Vision cameras currently use just a single exposure mode in all working mode.

The supported exposure mode is *Timed*, which means that the exposure time is fully controlled by features *Exposure Time* and *Automatic Exposure*, both discussed above. In particular, in the triggered mode, the trigger width has no impact on the exposure time.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// optionally set the exposure mode
// ('Timed' is anyway currently the only, thus default, option)
LvSetEnum (hDev, LvDevice_ExposureMode, LvExposureMode_Timed);
```

GEN<i>CAM

```
// optionally set the exposure mode
// ('Timed' is anyway currently the only, thus default, option)
write_enum (ExposureMode, Timed)
```

On an older series of PicSight-GigE cameras, "Trigger Width" exposure mode was used when triggering over the TTL input. In this mode, the exposure time was controlled by the width of the trigger signal. These legacy cameras are, however, obsolete, and the "Trigger Width" more is no more available on current camera series.

6.6.3. Trigger modes

The following sections give an overview of the trigger modes supported by the cameras. Detailed timing specifications are model specific.

6.6.3.1. Free running mode

In the free running mode no external triggers are considered by the camera. A continuous video stream with constant frame rate (see Appendix A, *Technical specifications for individual models* [p. 239]) is output by the camera. Exposure time is programmable.

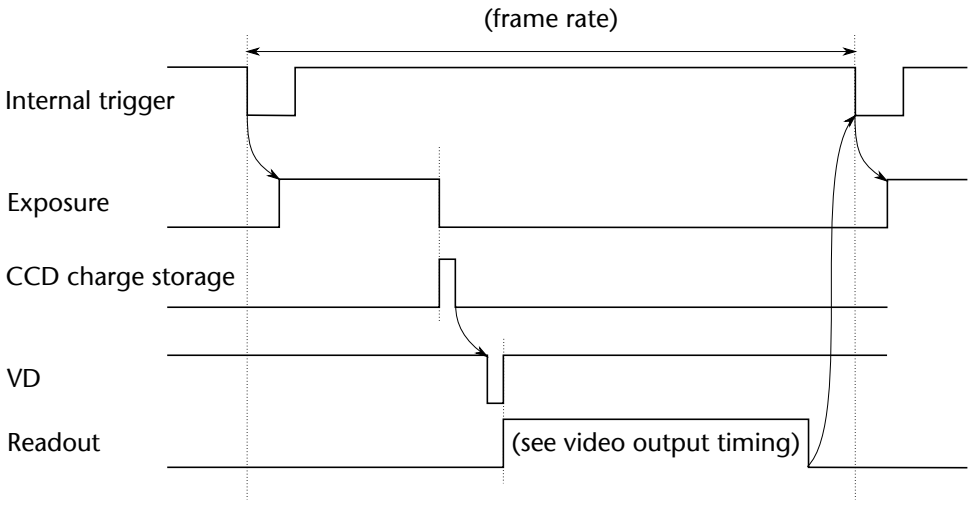


Figure 6.17. Free running mode timing

6.6.3.2. Triggered mode (accurate exposure)

In the triggered mode only the leading edge of the trigger pulse is significant. When encountered an exposure starts, eventually after a programmable delay. The length of the exposure is programmable.

The trigger should be delivered through one of the input lines (see Section 3.8.3, “Trigger input” [p. 33], Section 3.9, “Connector and cable description” [p. 37]). The trigger pulse duration should be at least 2 H. Trigger overlapping is not supported, therefore the maximum frame rate in triggered mode is always lower than the frame rate in free running mode.

The programmable exposure time range is limited, but the exposure timing is accurate (no jitter) across the entire range.

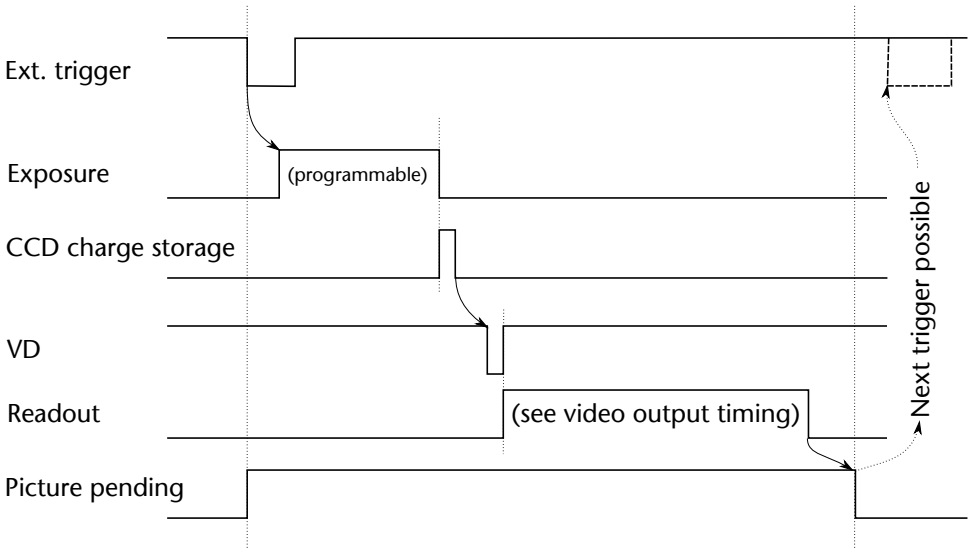


Figure 6.18. Triggered mode timing

6.6.3.3. Triggered mode (long range exposure)

Long range exposure mode is an alternative to the accurate exposure mode. The two modes are functionally equivalent, the long range exposure mode differs only in two aspects:

- The exposure time range is extended towards long time exposures.
- The short exposure times might be affected by a jitter.

6.6.3.4. Trigger modes of the CMOS rolling shutter cameras

The way how the pixel integration and readout is handled by rolling shutter cameras (see Section 3.6.1.3, “CMOS: global vs. rolling shutter” [p. 30]) is significantly different from cameras featuring global shutter. Thus also the trigger modes implemented by rolling shutter cameras are different.

Snapshot mode

Default trigger mode for the rolling shutter cameras.

In the snapshot mode only the leading edge of the trigger pulse is significant. When encountered, first row of the sensor is reset and its exposure starts. With a small offset, the second row and gradually all other rows are reset and their exposure starts. When the last row is reset, a period when all row are exposing starts. The possible (recommended) strobe illumination should fall into this period. When the common exposure time (strobe period) ends, first line is read out, then the second line and all other lines until entire sensor is read out.

Exposure time (meaning the “strobe” period when all rows are integrating) can be programmed in a sensor specific rang.

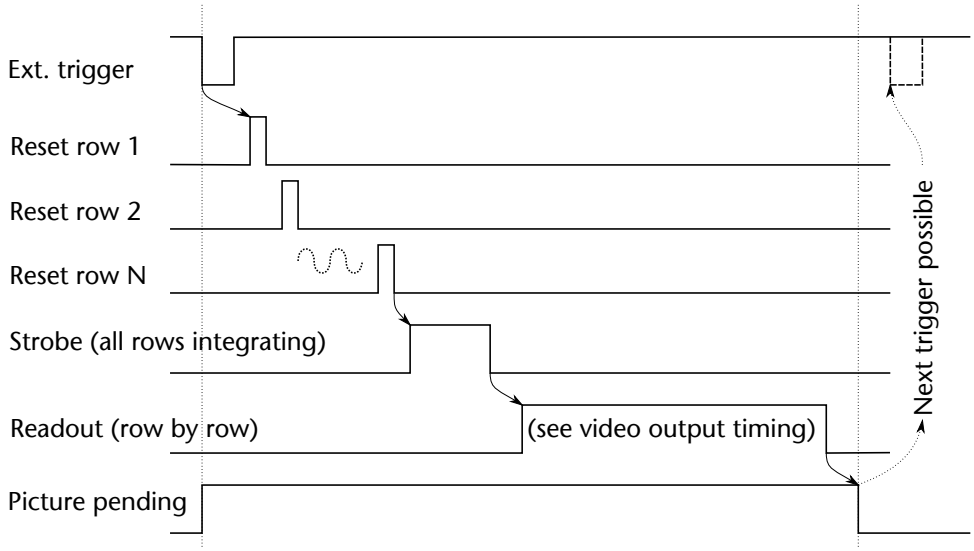


Figure 6.19. Snapshot mode timing

Global reset mode

Some rolling shutter sensors support additionally the global reset mode, which is very similar to the snapshot mode. The main difference is that all sensor rows are “globally” reset at the same time and the common integration time (strobe period) begins. After that, row-by-row sensor readout starts.

Because in global reset mode all the image rows are reset at the same time, but the readout is performed gradually, row-by-row, as usual for rolling shutter sensors, the exposure time is actually different for every image row (shortest for the first, longest for the last row). Thus it is highly recommended to use this mode only with a strobe illumination falling into the common exposure time period.

The main advantage over the snapshot mode is that the global reset mode is significantly faster (with the snapshot mode you loose one frame time between the actual trigger and start of the strobe period).

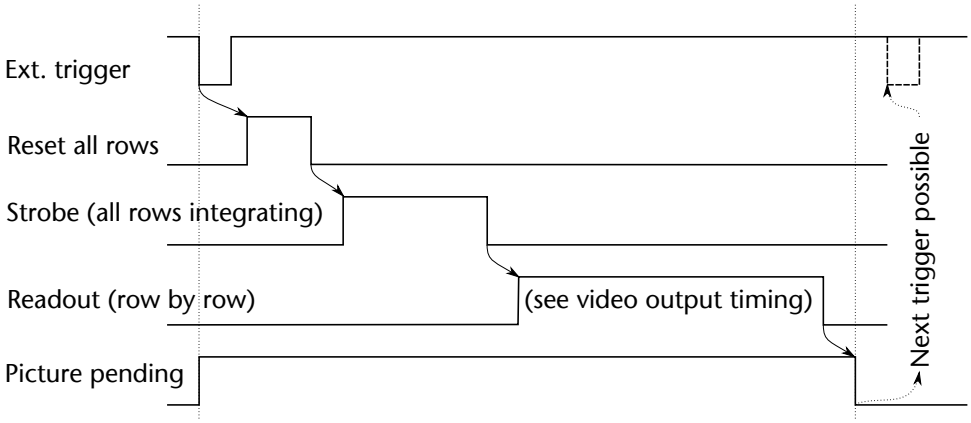


Figure 6.20. Global reset mode timing

6.6.4. Trigger configuration

When configuring the acquisition trigger, it's first necessary to select the trigger type to be configured. The camera currently supports just a single type (*Frame Start* trigger), but more trigger types will likely be added in future firmware versions. After switching the trigger on, the additional trigger options will become available. Note that the trigger configuration cannot be changed during acquisition.

Following steps document the procedure to configure the acquisition trigger:

1. **Select trigger type.** Use the *Trigger Selector* feature to select the trigger type to be configured. In current firmware version, only the *Frame Start* trigger is available. Each occurrence of this trigger starts acquisition of a new image frame.

2. **Switch the trigger on/off.** Set the *Trigger Mode* feature *On*/*Off*. When switched off, the camera operates in free running mode, no additional trigger parameters are available. When switched on, the camera operates in triggered mode and other trigger parameters can be configured.

3. **Specific trigger modes.** If a specific trigger mode should be used, switch on corresponding flag (*Long Range Exposure Mode* or *Global Reset Mode*).

4. **Trigger source (input line).** Configure the source of the trigger signals (*Trigger Source* feature). It can be a physical input line, such as *Line 1* (see Section 6.7, “Digital I/O, counters and timers” [p. 113], Section 3.8, “Input and output signals” [p. 32], Section 3.9, “Connector and cable description” [p. 37]). Alternatively, the trigger source can be set to *Software Trigger* — then the triggers are generated by the application, executing the command feature *Generate Software Trigger*. Note that while using a physical line as trigger source, the trigger can still be forced through the *Generate Software Trigger* command.

   Finally, the trigger source can be connected to one of the action commands, such as *Action 1*, if supported by the camera (see Section 6.8, “Action commands control” [p. 125]).

5. **Trigger activation (polarity).** The *Trigger Activation* feature controls which edge of the trigger input signal will be active, ie. which edge will actually ignite the trigger action. Possible values are *Rising Edge* and *Falling Edge*, rising edge being the transition of signal level from logical `0` to logical `1`. The interpretation of signal levels for different input types (TTL/opto-

coupler) is discussed in Section 3.8, “Input and output signals” [p. 32]. Not applicable for software triggering.

6. **Other options.** Eventually other fine-tuning options can be adjusted. Thes include:

   - *Trigger Delay* — delay in µs between the trigger and actual exposure start
   - *Trigger Divider* — allows to filter the trigger signals by a given ratio, only every Nth trigger is used
   - *Trigger Caching* — decides what to do, if a trigger arrives too early, before it can be accepted by the camera. Possible values are *Drop* (the trigger is ignored) and *Cache* (the trigger is applied as soon as the camera is ready).

7. **Triggering.** The triggers, being generated from external device using a trigger input line or from the application through the *Generate Software Trigger* command, can be issued as soon as the acquisition is started. Eventual physical triggers delivered when no acquisition is active will be ignored, the *Generate Software Trigger* command will not be executable.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// configure triggered mode
LvSetEnum (hDev, LvDevice_TriggerSelector, LvTriggerSelector_FrameStart);
LvSetEnum (hDev, LvDevice_TriggerMode, LvTriggerMode_On);
// optionally use a specific trigger mode, if supported
if (LvIsAvailable (hDev, LvDevice_LvLongRangeExposureMode))
{
    LvSetBool (hDev, LvDevice_LvLongRangeExposureMode, 1);
}
if (LvIsAvailable (hDev, LvDevice_LvGlobalResetMode))
{
    LvSetBool (hDev, LvDevice_LvGlobalResetMode, 1);
}
// configure hardware trigger input
LvSetEnum (hDev, LvDevice_TriggerSource,
                 LvTriggerSource_Line1); // opto input
LvSetEnum (hDev, LvDevice_TriggerActivation,
                 LvTriggerActivation_RisingEdge);
// configure software trigger input
LvSetEnum (hDev, LvDevice_TriggerSource, LvTriggerSource_Software);
// configure trigger over an action command
LvSetEnum (hDev, LvDevice_TriggerSource, LvTriggerSource_Action1);
// other options
LvSetFloat (hDev, LvDevice_TriggerDelay, 100.0); // 100 µs
LvSetEnum (hDev, LvDevice_LvTriggerCaching,
                 LvTriggerCaching_Cache); // or 'LvTriggerCaching_Drop'
// issue software trigger command after starting the acquisition
// - to generate regular trigger in software trigger mode
// - to force the trigger in hardware trigger mode
LvCmdExecute (hDev, LvDevice_TriggerSoftware);
// switch back to free-running mode
LvSetEnum (hDev, LvDevice_TriggerSelector, LvTriggerSelector_FrameStart);
LvSetEnum (hDev, LvDevice_TriggerMode, LvTriggerMode_Off);
```

GEN<i>CAM

```
// configure triggered mode
write_enum (TriggerSelector, FrameStart)
write_enum (TriggerMode, On)
// optionally use a specific trigger mode, if supported
write_enum (LvLongRangeExposureMode, true)
write_enum (LvGlobalResetMode, true)
// configure hardware trigger input
write_enum (TriggerSource, Line1) // opto input
write_enum (TriggerActivation, RisingEdge)
// configure software trigger input
write_enum (TriggerSource, Software)
// configure trigger over an action command
write_enum (TriggerSource, Action1)
// other options
write_float (TriggerDelay, 100.0) // 100 µs
write_enum (LvTriggerCaching, Cache) // or 'Drop'
// issue software trigger command after starting the acquisition
// - to generate regular trigger in software trigger mode
// - to force the trigger in hardware trigger mode
exec_cmd (TriggerSoftware)
// switch back to free-running mode
write_enum (TriggerSelector, FrameStart)
write_enum (TriggerMode, Off)
```

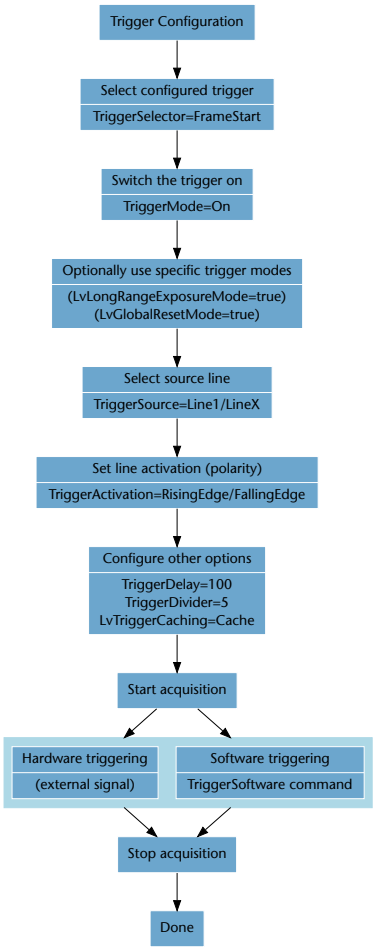


Figure 6.21. Flow chart: trigger mode configuration

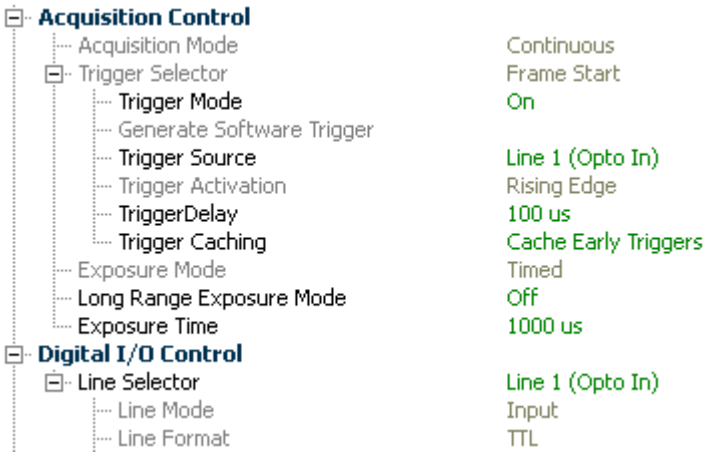


Figure 6.22. Example trigger configuration: triggering over Line1 (optocoupler input), activation on rising edge, trigger delay 100 µs

6.6.5. Frame rate

The frame rate (frequency of acquired image frames) of course varies among camera models, but it also depends on current settings, in particular:

- Pixel clock (sensor digitization clock)
- Area of interest (AOI, also known as “partial scan”)
- Binning/decimation

In triggered mode, the actual frame rate obviously depends on the frequency of incoming triggers, but it still cannot exceed the maximum defined by the current settings. The maximum achievable frame rate in triggered mode is lower than in the free running mode, because in free running mode the exposure of a new frame overlaps with readout of the previous one, while in the triggered mode the two operations are sequential (new trigger is not accepted while readout of the previous frame is still in progress).

When using the camera in the free-running mode (non-triggered), it is sometimes useful to decrease the frame rate from it's maximal value to a desired lower one. The reasons might be to moderate the processing load or network load when full frame rate is not needed.

To enable the frame rate settings, the *Acquisition Frame Rate Control* mode must be enabled first (switched to *On*). When set to this mode, the camera internally switches to triggered mode and generates periodic internal triggers with the desired frequency. The *Acquisition Frame Rate* feature itself is a floating point value in Hz.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// configure frame rate in Hz
// (available only when operation in non-triggered mode)
if (LvIsAvailable (hDev, LvDevice_LvAcquisitionFrameRateControlMode))
{
    LvSetEnum (hDev, LvDevice_LvAcquisitionFrameRateControlMode,
                     LvAcquisitionFrameRateControlMode_On);
    LvSetFloat (hDev, LvDevice_AcquisitionFrameRate, 20.0);
}
// switch back to regular free-running mode (max frame rate)
LvSetEnum (hDev, LvDevice_LvAcquisitionFrameRateControlMode,
                     LvAcquisitionFrameRateControlMode_Off);
```

GEN<i>CAM

```
// configure frame rate in Hz
// (available only when operation in non-triggered mode)
write_enum (LvAcquisitionFrameRateControlMode, On)
write_float (AcquisitionFrameRate, 20.0)
// switch back to regular free-running mode (max frame rate)
write_enum (LvAcquisitionFrameRateControlMode, Off)
```

Since camera internally switches to triggered mode for frame rate control, the maximum achievable frame rate is lower than the native free-running mode rate (when frame rate control mode is off).

The frame rate feature can never increase the frequency beyond the maximum defined by current working mode (see above), it can only decrease it. To reach a higher frame rate, adjusting corresponding settings (such as AOI or pixel clock) should be considered.

6.7. Digital I/O, counters and timers

This section discusses the digital I/O signal configuration.

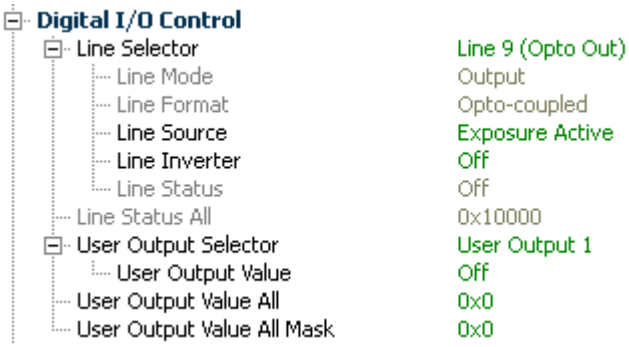


Figure 6.23. Example GUI showing the digital I/O features

Figure 6.24. Example GUI showing the counters and timers features

6.7.1. Camera I/O lines

The number and type of I/O lines might differ among camera models. The overview of the lines available and their assignment to actual physical pins on external connectors is given at Section 3.8, “Input and output signals” [p. 32] and Section 3.9, “Connector and cable description” [p. 37].

To query the information about camera's I/O lines, one may loop through the *Line Selector* feature and read the line mode (input/output) and format (optocoupler/TTL). In general, the interface

recognizes lines `Line1` through `Line32`, but only a subset of them might be implemented by a camera model.

The line mapping is fixed for all our camera families — all PicSight and CheckSight models (the legacy firmware versions 1.x of PicSight-GigE used different mapping). It's briefly summarized in the following table, details are listed in Section 3.8, “Input and output signals” [p. 32] and Section 3.9, “Connector and cable description” [p. 37].

I/O lines	Mapped signals
`Line1`—`Line4`	Optocoupler inputs
`Line5`—`Line8`	Reserved
`Line9`—`Line12`	Optocoupler outputs
`Line13`—`Line16`	Reserved
`Line17`—`Line18`	TTL inputs
`Line19`—`Line24`	Reserved
`Line25`—`Line26`	TTL outputs
`Line27`—`Line32`	Reserved

Table 6.1. Overview of digital I/O line mapping

Simplon

```
LvHDevice hDev; // open the device to get the handle
// query info about line 4
if (LvIsAvailableEnumEntry(hDev, LvDevice_LineSelector,
                                 LvLineSelector_Line4))
{
    LvSetEnum (hDev, LvDevice_LineSelector, LvLineSelector_Line4);
    int32_t EnumVal;
    // line mode gives LvLineMode_Output or LvLineMode_Input
    LvGetEnum (hDev, LvDevice_LineMode, &EnumVal);
    // line format gives LvLineFormat_OptoCoupled or LvLineFormat_TTL
    LvGetEnum (hDev, LvDevice_LineFormat, &EnumVal);
}
```

GEN<i>CAM

```
// query info about line 4
write_enum (LineSelector, Line4)
read_enum (LineMode) // Input or Output
read_enum (LineFormat) // OptoCoupled or TTL
```

6.7.1.1. Physical and logical lines

Except of the actual “physical” signals (representing the electrical status of the individual input/output lines, possibly inverted by the *Line Inverter* flag), few additional “logical” signals can also be used for configuration. These are virtual signals representing internal status of the camera. The available logical signals are:

- *Exposure Active* (`ExposureActive`) — the signal is asserted during an active exposure, both start and duration of the signal is aligned with the exposure
- *Frame Trigger* (`FrameTrigger`) — asserted, when a frame trigger is issued, no matter if issued by an external event, software command or other means. Only the starting edge of the signal

is significant, the duration is undefined. The signal is asserted immediately when the frame trigger arrives, before applying the *Trigger Delay*.
- *Timer X Active* (`TimerXActive`) — aligned with active period of a timer. The signal starts as soon as the *Timer Delay* expires and stays active during entire *Timer Duration*. Discussed below.
- *Counter X End* (`CounterXEnc`) — asserted when a counter expires (reaches *Counter Duration*). Only the starting edge is significant, duration of this signal is undefined. Discussed below.
- *User Output X* (`UserOutputX`) — asserted as long as given user output is set. Only the starting edge is significant, duration of this signal is undefined. Discussed below.

6.7.2. Observing I/O lines' status

There are two ways to query the current status of the camera's I/O lines:

- Status of the (single) line currently selected in the *Line Selector* is reported in the boolean feature *Line Status*. Note that the status of the output lines can be inverted using *Line Inverter* flag, as will be described shortly below. The way, how the signal levels of individual I/O types (TTL/optocoupler) are interpreted as logical `0` and `1` is described in Section 3.8, “Input and output signals” [p. 32].

- Status of all the I/O lines can be querying in a single access using *Line Status All* integer. This feature is a 32-bit bitfield each bit representing status of a single line. The least significant bit corresponds with *Line 1*, next bit with *Line 2* etc.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// read current status of line 2
LvSetEnum (hDev, LvDevice_LineSelector, LvLineSelector_Line2);
int32_t IntVal;
LvGetBool (hDev, LvDevice_LineStatus, &IntVal);
// read status of all lines as a bitfield in single access
LvGetInt32 (hDev, LvDevice_LineStatusAll, &IntVal);
```

GEN<i>CAM

```
// read current status of line 2
write_enum (LineSelector, Line2)
read_bool (LineStatus)
// read status of all lines as a bitfield in single access
read_int (LineStatusAll)
```

The status of I/O lines associated with individual acquired images can also be monitored through the chunk data, see Section 6.11, “Chunk data” [p. 146].

6.7.3. Using input lines for triggering

The signals issued on the input lines can be used for triggering the acquisition and other actions. The trigger input lines are not configured, however, through features in the Digital I/O category, but through features of the respective subsystem. Refer to

- *Trigger Source* feature in Section 6.6, “Acquisition and trigger control” [p. 101] to see how to configure acquisition trigger input line.
- *Special Purpose Trigger Source* feature in Section 6.15, “Special features” [p. 163].

6.7.4. Basic output (strobe) configuration

The most typical use case for digital output is possibly driving a strobe light. The simple case of strobe aligned with active exposure can be configured easily — select the desired line in *Line Selector* and set its *Line Source* to *Exposure Active* logical signal. If desired, the output signal polarity can be inverted.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// configure strobe output aligned withe exposure on line 10
// (remeber to configure triggered mode first)
LvSetEnum (hDev, LvDevice_LineSelector, LvLineSelector_Line10);
LvSetEnum (hDev, LvDevice_LineSource, LvLineSource_ExposureActive);
// optionally invert the output signal polarity
LvSetBool (hDev, LvDevice_LineInverter, 1);
```

GEN<i>CAM

```
// configure strobe output aligned withe exposure on line 10
// (remeber to configure triggered mode first)
write_enum (LineSelector, Line10)
write_enum (LineSource, ExposureActive)
// optionally invert the output signal polarity
write_bool (LineInverter, true)
```

Figure 6.25. Simple strobe configuration (strobe aligned with exposure)

It is possible to drive multiple output lines by the same (eg. *Exposure Active*) signal at the same time. To disconnect an output line again, you simply set its line source back to *Off*.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// disconnect the output signal on line 10
LvSetEnum (hDev, LvDevice_LineSelector, LvLineSelector_Line10);
LvSetEnum (hDev, LvDevice_LineSource, LvLineSource_Off);
```

GEN<i>CAM

```
// disconnect the output signal on line 10
write_enum (LineSelector, Line10)
write_enum (LineSource, Off)
```

Figure 6.26. Output line disconnected

6.7.5. User output configuration

As an alternative to the digital line output configuration synchronized with a cameras internal signals (such as “exposure active” signal described above), the status of the individual output lines can be directly controlled by user actions. This functionality is implemented by the “user output” features in the *Digital I/O Control* category.

To directly control status of an output line, select the line and set its *Line Source* to one of the user outputs, eg. *User Output 1*. Multiple lines can be driven by the same user output. Unlike with the *Exposure Active* signal, the user outputs ara always available, even in the free running mode.

Once the configuration is finished, the output can be driven by toggling value of the corresponding user output — select the connected output in *User Output Selector*, then set its value through the *User Output Value* boolean feature.

The *Line Inverter* functionality still applies as usual for lines controlled through user outputs.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// configure line output on line 10 to be based on user output 1
LvSetEnum (hDev, LvDevice_LineSelector, LvLineSelector_Line10);
LvSetEnum (hDev, LvDevice_LineSource, LvLineSource_UserOutput1);
// now toggle the line status through the user output value as needed
LvSetEnum (hDev, LvDevice_UserOutputSelector,
                 LvUserOutputSelector_UserOutput1);
LvSetBool (hDev, LvDevice_UserOutputValue, 1);
LvSetBool (hDev, LvDevice_UserOutputValue, 0);
```

GEN<i>CAM

```
// configure line output on line 10 to be based on user output 1
write_enum (LineSelector, Line10)
write_enum (LineSource, UserOutput1)
// now toggle the line status through the user output value as needed
write_enum (UserOutputSelector, UserOutput1)
write_bool (UserOutputValue, true)
write_bool (UserOutputValue, false)
```

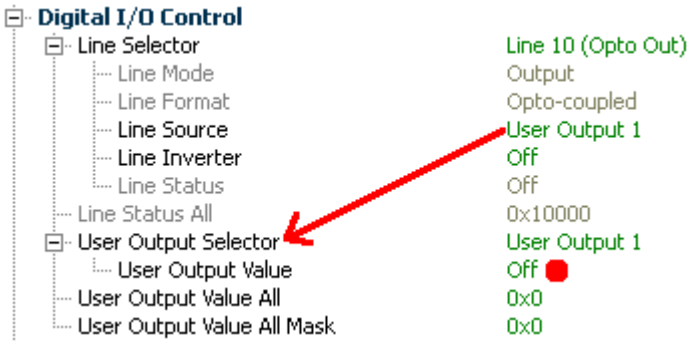


Figure 6.27. Line output (strobe) controlled through the user output settings

When working with multiple user outputs, it is possible to adjust them all in a single access. The camera provides two integer features, *User Output Value All* and *User Output Value All Mask*, both 32-bit bitfields. The least significant bit of the bitfields corresponds with *User Output 1*, the next bit with *User Output 2* etc.

The *User Output Value All Mask* bitfield defines the set of the user outputs to be adjusted, *User Output Value All* defines the logical values for them. When writing a new *User Output Value All* value, the camera considers only those user outputs being enabled (having "their" bit set) in the mask. The user outputs disabled in the mask (bit cleared) are preserved — their status is left intact, regardless the *User Output Value All* value.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// (connect line sources to user outputs as desired...)
// set the 'value-all-mask' to enable user output 1 and 2
LvSetInt32 (hDev, LvDevice_UserOutputValueAllMask, 0x3);
// following call sets just the enabled outputs to logical 1
// the other ones are left intact, keeping their current status
LvSetInt32 (hDev, LvDevice_UserOutputValueAll, 0xFF);
// next call sets output 1 to logical 1, outut 2 to logical 0
// the other ones are left intact, keeping their current status
LvSetInt32 (hDev, LvDevice_UserOutputValueAll, 0x1);
```

GEN<i>CAM

```
// (connect line sources to user outputs as desired...)
// set the 'value-all-mask' to enable user output 1 and 2
write_int (UserOutputValueAllMask, 0x3)
// following call sets just the enabled outputs to logical 1
// the other ones are left intact, keeping their current status
write_int (UserOutputValueAll, 0xFF)
// next call sets output 1 to logical 1, outut 2 to logical 0
// the other ones are left intact, keeping their current status
write_int (UserOutputValueAll, 0x1)
```

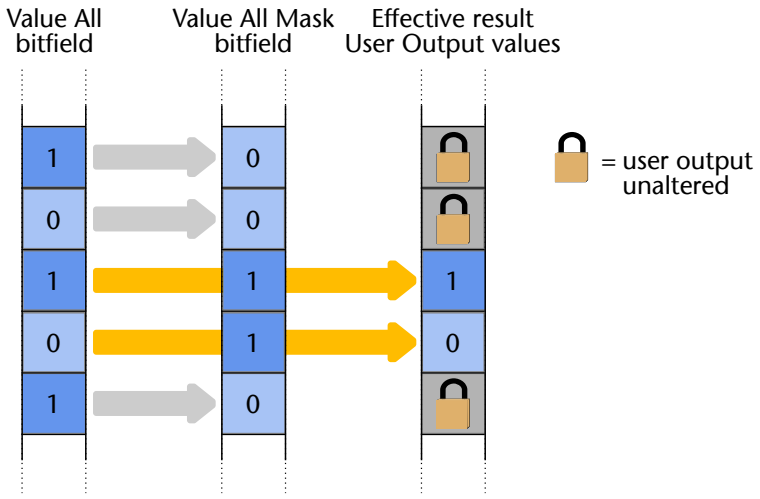


Figure 6.28. User Output Value All functionality illustrated

6.7.6. Advanced output configuration

Sometimes the digital output timing has to be different than the exposure timing. In such case, the output can be driven by a timer, started by the *Frame Trigger* event. The timer has a configurable *Timer Delay* and *Timer Duration* (both in µs). The delay defines, how long after receiving the frame acquisition trigger the camera asserts the output. The duration defines, how long stays the output active. The *Line Source* needs than be connected to such configured timer. To keep the description simple, let's describe the desired settings in source code and screenshot of finished configuration in Simplon Explorer.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// configure strobe output on line 10,
// starting 100 µs after frame trigger and staying active 1000 µs
// first configure the timer
LvSetEnum (hDev, LvDevice_TimerSelector, LvTimerSelector_Timer1);
LvSetEnum (hDev, LvDevice_TimerTriggerSource,
                 LvTimerTriggerSource_FrameTrigger);
LvSetFloat (hDev, LvDevice_TimerDelay, 100.0);
LvSetFloat (hDev, LvDevice_TimerDuration, 1000.0);
// now connect the line source to the configured timer
LvSetEnum (hDev, LvDevice_LineSelector, LvLineSelector_Line10);
LvSetEnum (hDev, LvDevice_LineSource, LvLineSource_Timer1Active);
// optionally invert the output signal polarity
LvSetBool (hDev, LvDevice_LineInverter, 1);
```

GEN<i>CAM

```
// configure strobe output on line 10,
// starting 100 µs after frame trigger and staying active 1000 µs
// first configure the timer
write_enum (TimerSelector, Timer1)
write_enum (TimerTriggerSource, FrameTrigger)
write_float (TimerDelay, 100.0)
write_float (TimerDuration, 1000.0)
// now connect the line source to the configured timer
write_enum (LineSelector, Line10)
```

```
write_enum (LineSource, Timer1Active)
// optionally invert the output signal polarity
write_bool (LineInverter, true)
```

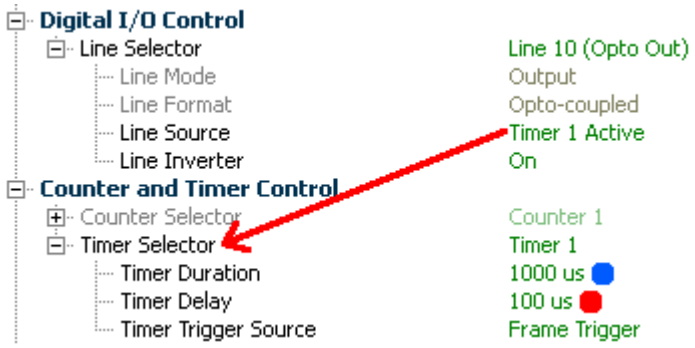


Figure 6.29. Strobe configured to be delayed 100 µs behind trigger and last 1000 µs

Note that any of the timers (*Timer 1* through *Timer 4*) could be used.

Using the described strobe timing configuration together with the *Trigger Delay* feature, the desired timing of the strobe and exposure itself can be precisely configured. Setting *Trigger Delay* larger than the strobe delay, the exposure starts later than the strobe signal itself.

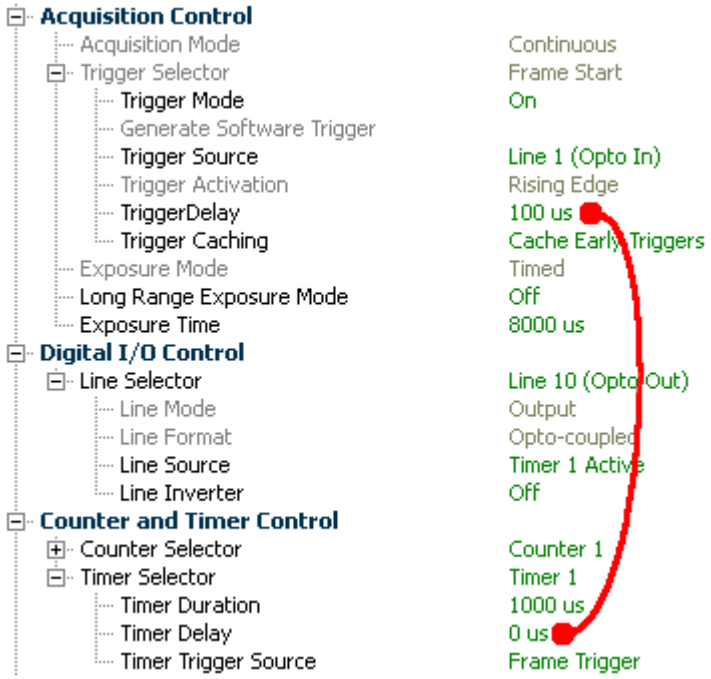


Figure 6.30. Strobe configured to be start 100 µs before exposure start

Sometimes it's desirable to assert the output just once in a number of events (frame triggers). This might be required in use cases such as inter-camera synchronization (see below) or similar. Such scenario is configured in the same way as the previous case (delayed output), just the timer is not connected directly to the frame trigger event, but through a counter. The timer (and any line output connected) would then be asserted only when the counter expires — the counter thus effectively behaves as signal divider. Let's have a look at the previous example modified to assert the output only at every fourth frame trigger event.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// configure strobe output on line 10,
// starting 100 µs after every 4th frame trigger
// and staying active 1000 µs
// first configure the counter
LvSetEnum (hDev, LvDevice_CounterSelector, LvCounterSelector_Counter1);
LvSetEnum (hDev, LvDevice_CounterEventSource,
                 LvCounterEventSource_FrameTrigger);
LvSetInt32 (hDev, LvDevice_CounterDuration, 4);
LvSetEnum (hDev, LvDevice_LvCounterMode,
                 LvCounterMode_Autoreset); // restarts when expired
// connect the timer and configure it
LvSetEnum (hDev, LvDevice_TimerSelector, LvTimerSelector_Timer1);
LvSetEnum (hDev, LvDevice_TimerTriggerSource,
                 LvTimerTriggerSource_Counter1End);
LvSetFloat (hDev, LvDevice_TimerDelay, 100.0);
LvSetFloat (hDev, LvDevice_TimerDuration, 1000.0);
// now connect the line source to the configured timer
LvSetEnum (hDev, LvDevice_LineSelector, LvLineSelector_Line10);
LvSetEnum (hDev, LvDevice_LineSource, LvLineSource_Timer1Active);
// optionally invert the output signal polarity
LvSetBool (hDev, LvDevice_LineInverter, 1);
```

GEN<i>CAM

```
// configure strobe output on line 10,
// starting 100 µs after every 4th frame trigger
// and staying active 1000 µs
// first configure the counter
write_enum (CounterSelector, Counter1)
write_enum (CounterEventSource, FrameTrigger)
write_int (CounterDuration, 4)
write_enum (CounterMode, Autoreset) // restarts when expired
// connect the timer and configure it
write_enum (TimerSelector, Timer1)
write_enum (TimerTriggerSource, Counter1End)
write_float (TimerDelay, 100.0)
write_float (TimerDuration, 1000.0)
// now connect the line source to the configured timer
write_enum (LineSelector, Line10)
write_enum (LineSource, Timer1Active)
// optionally invert the output signal polarity
write_bool (LineInverter, true)
```

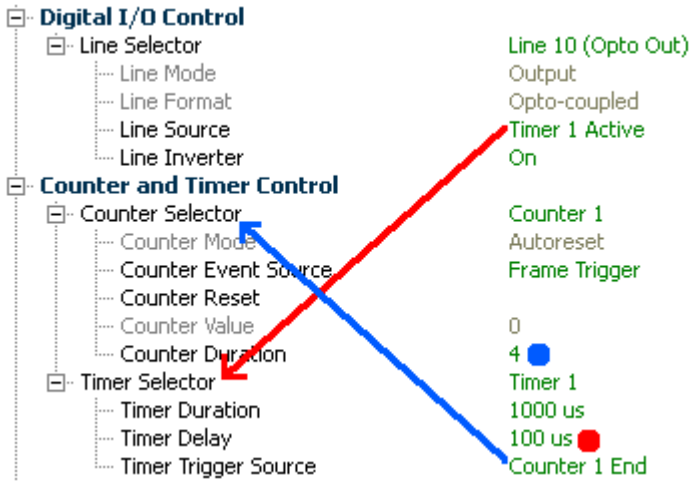


Figure 6.31. Output configured to be fired on every fourth trigger (an delayed 100 µs)

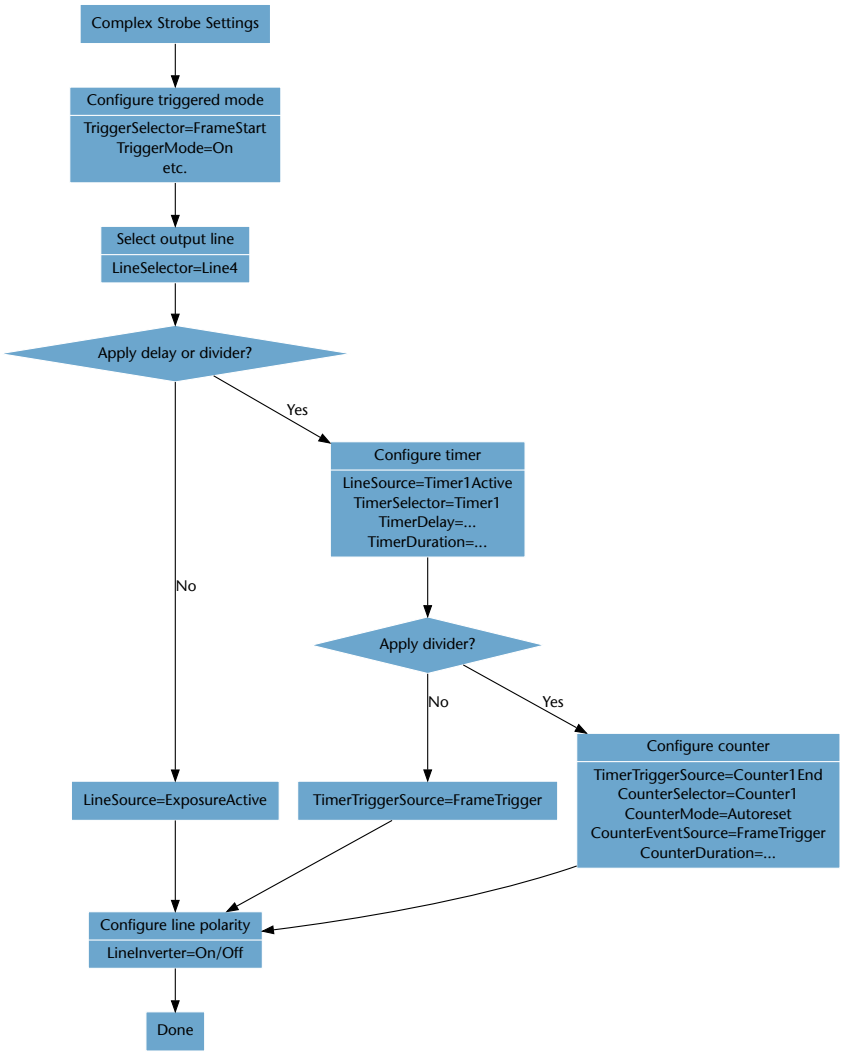


Figure 6.32. Flow chart: strobe configuration options

6.7.7. Disconnected outputs

When an output line is disconnected (its line source is set to *Off*), its status is always switched to its default (high impedance for optocoupler, driven high for TTL), regardless the actual value of the line inverter flag. Refer also to Section 3.8, “Input and output signals” [p. 32].

The line might be effectively disconnected even if the line source is not Off, for example if the line source is set to Timer 1 Active, but the timer itself is disconnected (not driven by any signal).

6.7.8. Use for inter-camera synchronization

One of possible use cases for the output signal “divider” described above is to synchronize frame ID's and timestamps across multiple cameras working together. If an application needs to pair images coming from multiple cameras, it needs to get them synchronized regularly. A possible solution to this problem is to output this synchronizing pulse once every “N” (eg. every 100) images are acquired on the master camera and route this signal to the timestamp-reset input on all involved cameras. The procedure to output a signal every Nth image was described above, the timestamp resetting triggers are discussed in Section 6.15, “Special features” [p. 163].

ACQ TRG: *acquisition trigger input*
TRG CPY: *output copying every Nth trigger*
RST TRG: *timestamp reset trigger input*

Figure 6.33. Using digital I/O to periodically synchronize timestamps among multiple cameras

6.7.9. Input lines debouncer

The camera's input lines (both TTL and optocoupler) are equipped with an input signal debouncer. The task of the debouncer is to detect the input signal changes, toggle the logical signal level once and wait for a configurable time period before sampling the input signal again. For input signals with slow (or fuzzy) rising or falling edges, the debounce logic is used to prevent unwanted toggling of the input signal after it has been first sampled.

The debouncer behaviour can be configured through a floating point feature *Line Debounce Duration*. This time (in µs) parameter specifies, for how long will the debounce logic avoid scanning the physical input again after detecting a level change. For optimal performance, the *Line Debounce Duration* should be set to a value such that the wait period is slightly longer than the worst-case rising or fall time of the input signal. Setting *Line Debounce Duration* to zero (`0`) will disable the debouncer.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// configure the debouncer on line 17,
// setting the debounce period duration to 5 µs
LvSetEnum (hDev, LvDevice_LineSelector, LvLineSelector_Line17);
LvSetFloat (hDev, LvLineDebounceDuration, 5.0);
// or disable the debouncer for the selected line
LvSetFloat (hDev, LvLineDebounceDuration, 0.0);
```

GEN<i>CAM

```
// configure the debouncer on line 17,
// setting the debounce period duration to 5 µs
write_enum (LineSelector, Line17)
write_float (LvLineDebounceDuration, 5.0)
// or disable the debouncer for the selected line
write_float (LvLineDebounceDuration, 0.0)
```

Note that this debouncing technique does not serve as an input signal noise filter, which would eliminate too short input pulses. The disadvantage of the noise filters is that they necessarily delay the input signal, waiting until it can be evaluated as valid. Thus an unwanted latency is added to the input. With our debouncer implementation, the input signal is asserted as soon as the signal (however short) arrives — the debouncer just disables the input scanning for a given period after input change is detected.

Besides the debouncer, the input circuit is also equipped with glitch detection logic, which ensures that even very fast input signal changes (under 1 µs) can be detected and processed by the digital input subsystem.

6.7.10. Special topics

Just to make the discussion complete, we have to mention last few special timer/counter related features. First is the ability to software reset the counter at any time (using *Counter Reset* command, so that it starts counting again from zero. Second is the option to adjust counter's current *Counter Value* to any number lower than the counter duration.

Note that the counter value is only reset to zero when the counter expires or when calling the *Counter Reset* command explicitly. Connecting/disconnecting the *Counter Event Source* or starting/stopping acquisition has no influence on the counter value, which is maintained across these actions. The application should reset the counter value explicitly whenever needed.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// reset Counter1 (it will start counting from zero again)
LvSetEnum (hDev, LvDevice_CounterSelector, LvCounterSelector_Counter1);
LvCmdExecute (hDev, LvDevice_CounterReset);
// set counter value to duration-1
// first time it will expire at the very first event
// next time it will count the full 'duration' cycle
LvSetEnum (hDev, LvDevice_CounterSelector, LvCounterSelector_Counter1);
LvSetInt32 (hDev, LvDevice_CounterDuration, 100);
LvSetInt32 (hDev, LvDevice_CounterValue, 99);
```

GEN<i>CAM

```
// reset Counter1 (it will start counting from zero again)
write_enum (CounterSelector, Counter1)
exec_cmd (CounterReset)
// set counter value to duration-1
// first time it will expire at the very first event
// next time it will count the full 'duration' cycle
write_enum (CounterSelector, Counter1)
write_int (CounterDuration, 100)
write_int (CounterValue, 99)
```

6.8. Action commands control

This section discusses the features related to the control of action commands optionally delivered to the camera(s).

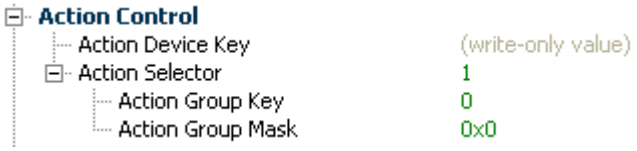


Figure 6.34. Example GUI showing the action control features

Action commands are intended to allow triggering various “actions” (typically a new frame acquisition) on multiple devices in parallel, in approximately same time. The commands are software generated “messages” delivered to the selected cameras through the Simplon GenTL Producer — or, in case of PicSight-GigE/PicSight-Smart from another GigE Vision compatible application with standard action command support. Note that because of their software nature (contrary to a hardware signal delivered over an I/O line), the action commands are not fully synchronous and may introduce certain jitter, dependent mainly on the actual system configuration.

The camera might recognize multiple different action commands. To find out the number of actions supported by the camera, read out the range of the *Action Selector* integer feature — it's minimum is always `1`, while the maximum corresponds with the number of recognized actions. When “connecting” to the individual actions, such as in *Trigger Source*, the actions are referred as *Action 1*, *Action 2*, etc.

Every action command generated by the application carries several parameters: “device key”, “group key” and “group mask”. The camera configuration contains corresponding matching features that are checked against the parameters carried by individual action commands delivered to the camera — to decide which (if any) actual action should be generated. This kind of fine grained matching is important, because the action commands are typically broadcast to all accessible devices. The camera's features controlling the action commands are:

Action Device Key

The action command will be only considered if the device is currently open by a controlling application and if the device key parameter included in the command equals the *Action Device Key* value configured in the device. If they do not match, the action command is immediately dropped without any further checks.

Note that the *Action Device Key* is a write-only 32-bit unsigned integer feature. It is write-only to ensure that the primary controlling application has full, exclusive control of the key. If the primary application desires to allow other applications (or even devices) to generate the action commands, it needs to pass the device key parameter to them.

Action Selector

Selects one of the action commands supported by the camera, allowing to configure the action-specific parameters, *Action Group Key* and *Action*

Group Mask. The selector is an integer, ranging from `1` to the maximum number of action commands recognized by the camera. The value `1` corresponds the action being referred as *Action 1*, value `2` corresponds with *Action 2* etc.

Action Group Key — If an action command arrives and passes the device key check, it's group key parameter is compared against the *Action Group Key* values (again 32-bit unsigned integers) configured for the individual actions. Those actions for which their group keys equal with the key embedded in the command are considered as candidates for generating the actual action commands.

Action Group Mask — The *Action Group Mask* is the last step in the action command matching chain. For the action(s) matching the group key parameter, the delivered group mask parameter is compared against the actions configured *Action Group Mask* value (yet another 32-bit unsigned integer. In this case, the camera performs the bit-wise AND operation on the two values. If the result is nonzero, the action is considered as a "match" and the corresponding internal action signal is generated. Any subsystems connected to that action (for example the *Trigger Source*) would immediately receive the signal.

Given this algorithm for action command delivery, it is solely choice of the application, how it uses the described parameters to control a group of cameras through the action commands. Note that many applications control these parameters "under the hood" and that they are all accessible only in the "guru" feature visibility level.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// set the action device key parameter
// (using LvSetInt64() since the int32 range might  not
// be sufficient for the unsigned 32-bit parameters)
LvSetInt64 (hDev, LvDevice_ActionDeviceKey, 0x07A312B2);
// configure the group key and mas for Action 1
LvSetInt32 (hDev, LvDevice_ActionSelector, 1);
LvSetInt64 (hDev, LvDevice_ActionGroupKey, 0x00000020);
LvSetInt64 (hDev, LvDevice_ActionGroupMask, 0x00000013);
```

GEN<i>CAM

```
// set the action device key parameter
write_int (ActionDeviceKey, 0x07A312B2)
// configure the group key and mas for Action 1
write_int (ActionSelector, 1)
write_int (ActionGroupKey, 0x00000020)
write_int (ActionGroupMask, 0x00000013)
```

6.8.1. Using action commands for triggering

The action commands issued from the application can be used for triggering the acquisition and other actions. The respective actions used to trigger the desired operations are not configured, however, through features in the Action Control category, but through features of the respective subsystem. Refer to

- *Trigger Source* feature in Section 6.6, "Acquisition and trigger control" [p. 101] to see how to configure acquisition trigger action command.
- *Special Purpose Trigger Source* feature in Section 6.15, "Special features" [p. 163].

6.8.2. Generating the action commands

The information how to generate and deliver the action commands to the desired cameras is beyond the scope of this section. It is responsibility of the controlling application. If you are programming with Simplon, refer to its documentation (Chapter 7, *Simplon GenTL Producer features* [p. 194], Simplon programmer's guide). When programming with another GigE Vision compatible library, find the relevant information in its documentation.

6.9. Analog controls

This section discusses the "analog" features of the image, ie. the features that affect the analog to digital conversion of the sensor output.

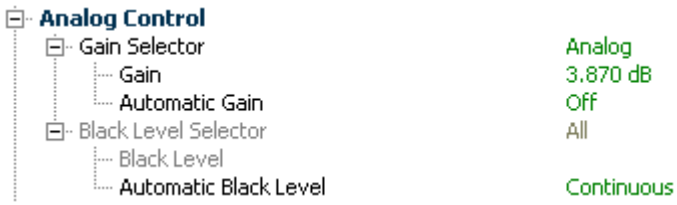


Figure 6.35. Example GUI showing the analog controls' features

6.9.1. Gain

Gain is a multiplicative factor applied to the pixel output values to increase the response — produce higher pixel values (higher brightness). It is useful when under given conditions (environment and maximal acceptable exposure time) the sensor does not reach the desired level of saturation. Note however, that together with the useful signal itself, gain amplifies also the noise, so it does not improve the real pixel dynamic range. It is therefore recommended when increasing the brightness, first adjust the exposure time (Section 6.6, "Acquisition and trigger control" [p. 101]) and only go for gain when exposure time can no more be increased.

There are two types of gain available (although not all camera modes support both of them):

Analog gain — Amplifies the actual analog signal output of the sensor, before actually digitizing it.

Digital gain — Amplifies already digitized pixel values (multiplying by a given factor). Since certain noise is also generated during the A/D conversion, digital gain also amplifies this additional noise and is thus less accurate than the analog gain.

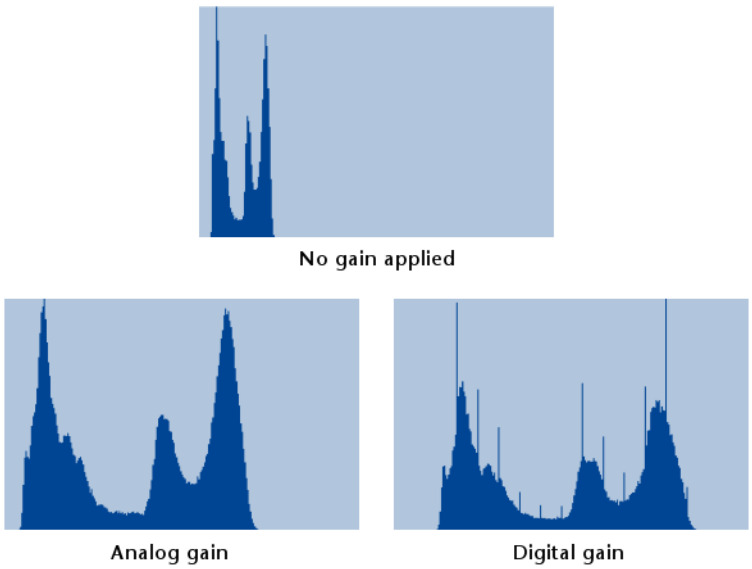


Figure 6.36. Comparison of analog and digital gain, the histogram shows how digital gain amplifies the digitization noise

Gain values are expressed in logarithmic unit, decibel (dB). Increasing the gain by 6 dB approximately doubles the camera response. To find correct gain value in dB for a desired pixel value multiplication factor, use formula: `Gain=20*log(MultiplicativeFactor)`.

The range of possible gain values is model specific, negative gain is also possible for certain models. The available range might differ in free running and individual triggered modes. Note that the camera might be able to adjust the gain only in small but discrete steps. It will align the requested value to the closest "step" — to know the exact used value, the application might want to re-read the gain after setting it.

All the camera models are adjusted, so that for gain set to 0 dB, the maximal sensor output (output of fully saturated sensor) produces the "white" pixels (pixel value 255 for 8-bit pixel format).

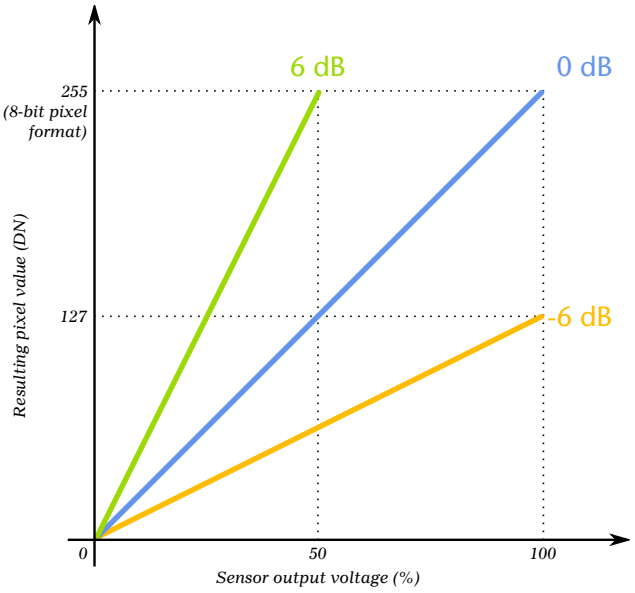


Figure 6.37. Output pixel values for various gains (assumes that no black level is applied)

Figure 6.38. Gain effect example (±6 dB)

Before reading/writing the *Gain* value, it is necessary to first select the gain type to be accessed in the *Gain Selector* enumeration. Possible values are *Analog* and *Digital* (although not all camera models support both of them).

Simplon

```
LvHDevice hDev; // open the device to get the handle
// adjust analog gain to 6 dB
// (assumes that it is not switched to automatic mode)
LvSetEnum (hDev, LvDevice_GainSelector, LvGainSelector_AnalogAll);
LvSetFloat (hDev, LvDevice_Gain, 6.0);
// if desired, re-read the value possibly adjusted by the camera
double FloatVal;
LvGetFloat (hDev, LvDev_Gain, &FloatVal);
// adjust digital gain (if available) to -6 dB
if (LvIsAvailableEnumEntry(hDev, LvDevice_GainSelector,
                                 LvGainSelector_DigitalAll))
{
    LvSetEnum (hDev, LvDevice_GainSelector, LvGainSelector_DigitalAll);
    LvSetFloat (hDev, LvDevice_Gain, -6.0);
}
```

GEN<i>CAM

```
// adjust analog gain to 6 dB
// (assumes that it is not switched to automatic mode)
write_enum (GainSelector, AnalogAll)
write_float (Gain, 6.0)
// adjust digital gain (if available) to -6 dB
```

```
write_enum (GainSelector, DigitalAll)
write_float (Gain, -6.0)
```

The camera might also adjust the gain automatically, see Section 6.9.3, “Automatic gain and black level” [p. 132].

On certain camera models (in particular those based on the G43A sensor), the gain is applied with a delay of one frame.

6.9.1.1. Default gain and exposure time settings

The factory defaults for gain and exposure time are adjusted for good “first test” experience, ie. to provide reasonable image brightness in regular office conditions in a typical illumination level, assuming lens aperture size 2.8. We assume that the user will have to adjust the camera anyway for the target working environment.

The order of adjusting the default values is as follows:

1. First try to adjust the image brightness by altering the exposure time. Do not, however, increase the exposure time beyond the value which would start affecting the camera's frame rate.
2. If the image is still too dark at the maximal exposure time level, finish the adjustment by increasing also the analog gain.
3. The digital gain, if available, is always kept at 0 dB.

6.9.2. Black level

The black level function (also known as “offset”) controls the output value of the black pixels, ie. pixels not exposed to illumination. It controls it by applying offset (positive or negative) to the individual pixel values. That offset is expressed as a floating point value in unit “digital number”, DN. Black level value 5 DN means that values of all pixels will increase approximately by 5. The offset is actually applied to the analog sensor output signal, before the analog to digital conversion.

The black level function can be used to keep the camera response constant in changing environment conditions. For example, if the camera and the sensor heats up, the response of the sensor will slightly increase thanks to the thermally generated pixel output. Decreasing the black level value will compensate for that. It is usually desirable to keep the values of the “black” pixels slightly above zero, so that a possible decrease of their value can also be recognized and appropriately compensated.

The range of possible values is model specific. Positive and negative values are possible, as well as decimal values. Note however, that the camera might be able to adjust the black level only in small but discrete steps. It will align the requested value to the closest “step” — to know the exact used value, the application might want to re-read the black level after setting it.

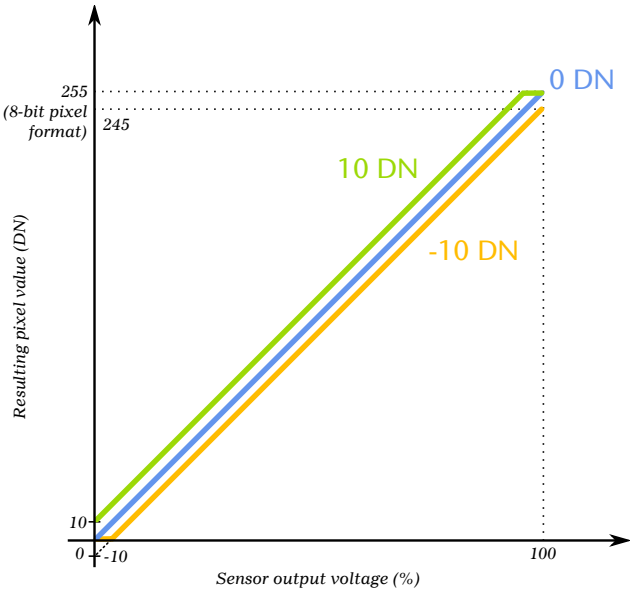


Figure 6.39. Output pixel values for various black levels (assumes that no gain is applied)

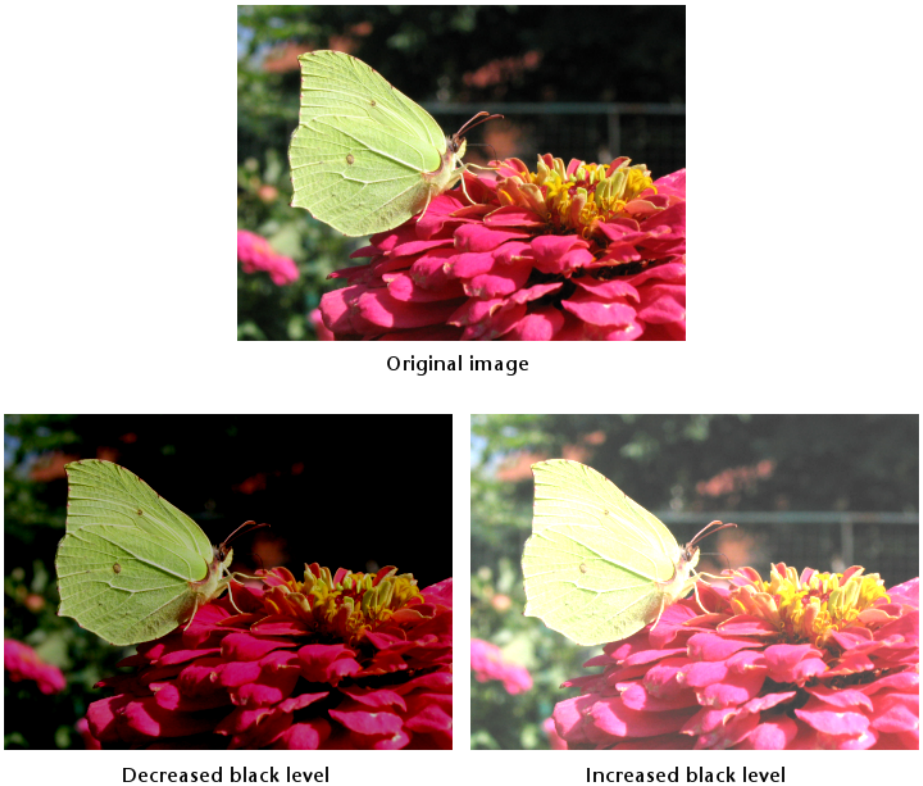


Figure 6.40. Black level example (artificially increased effect)

The function is controlled through the *Black Level* floating point feature. Since the cameras support just a single black level type, adjusting the *Black Level Selector* in the examples below is currently more less formal and could be omitted.

```
LvHDevice hDev; // open the device to get the handle
// adjust black level to 10.0 DN
// (assumes that black level is not switched to automatic mode)
LvSetEnum (hDev, LvDevice_BlackLevelSelector, LvBlackLevelSelector_All);
LvSetFloat (hDev, LvDevice_BlackLevel, 10.0);
```

GEN<i>CAM

```
// adjust black level to 10.0 DN
// (assumes that black level is not switched to automatic mode)
write_enum (BlackLevelSelector, All)
write_float (BlackLevel, 10.0)
```

Black level might also be adjusted automatically, see Section 6.9.3, “Automatic gain and black level” [p. 132].

6.9.2.1. Black level and gain

When using together with gain, the black level is applied first. The sensor analog output is first adjusted according to the black level settings and the result is then subject to amplification by the selected gain factor. The effect may be symbolically expressed using an equation (of course considering that gain has in fact logarithmic effect):
`OutputValue = (InputValue+BlackLevel) * 10^(Gain/20)`, see also Section 6.9.1, “Gain” [p. 127].

6.9.2.2. Black level and pixel format

Since the black level values are expressed in the “digital number” (DN) units, the same value can have different meaning for different pixel format. To reach the same effect, higher black level value is needed for pixel formats with more bits per pixel.

To give an example, a sensor pixel, which would give value 10 in 8-bit pixel format, would in 12-bit pixel format yield value around 160. Therefore also the black level value 10 DN in 8-bit pixel format has the same effect as 160 DN in 12-bit pixel format.

To keep the black level effect constant when changing the pixel format, the camera readjusts the black level range and the value itself whenever a pixel format is changed.

6.9.3. Automatic gain and black level

Similarly as automatic exposure (Section 6.6.2.1, “Automatic exposure” [p. 105]), selected camera models offer also automatic gain and automatic black level operation modes.

The automatic gain algorithm observes the luminance of acquired images and attempts to keep it at constant level, by compensating for any changes by adjusting the gain for subsequent frames. To prevent flickering, the adjustments are low-pass fltered — it may take several frames until the target luminance is restored again.

The automatic black level algorithm monitors the “dark pixels”, ie. the pixels out of the effective sensor area, which are not exposed to illumination. The value of the dark pixels is averaged over multiple acquired frames, low-pass filtered (to avoid fast oscillations) and compared to predefined thresholds. When the value falls outside those threshold, the black level is re-adjusted to restore the desired black values (very slightly above zero).

The configuration is analogical for all automatic features — each of them is controlled through an enumerated feature (*Automatic Gain*, *Automatic Black Level*, *Automatic Exposure*), which can switch between two modes, *Off* and *Continuous*. The *Off* option means the automatic mode is not active and given feature is controlled “manually” by adjusting the desired value. This is default

option for all the automatic features. The *Continuous* option means that the automatic adjustment of the feature is continuously applied.

To configure automatic gain, it is first necessary to select through *Gain Selector*, which gain type (analog or digital) is being configured.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// switch analog gain to automatic mode
// (full availability checking for generic applications)
if (LvIsAvailableEnumEntry(hDev, LvDevice_GainSelector,
                                 LvGainSelector_AnalogAll))
{
    LvSetEnum (hDev, LvDevice_GainSelector, LvGainSelector_AnalogAll);
    if (LvIsAvailableEnumEntry(hDev, LvDevice_GainAuto,
                                     LvGainAuto_Continuous))
    {
        LvSetEnum (hDev, LvDevice_GainAuto, LvGainAuto_Continuous);
    }
}
// switch digital gain to automatic mode
// (since now the extended checks omitted)
LvSetEnum (hDev, LvDevice_GainSelector, LvGainSelector_DigitalAll);
LvSetEnum (hDev, LvDevice_GainAuto, LvGainAuto_Continuous);
// switch to automatic black level mode
LvSetEnum (hDev, LvDevice_BlackLevelSelector, LvBlackLevelSelector_All);
LvSetEnum (hDev, LvDevice_BlackLevelAuto, LvBlackLevelAuto_Continuous);
// switch the analog gain back to manual mode,
// reading the newly adjusted value
LvSetEnum (hDev, LvDevice_GainSelector, LvGainSelector_AnalogAll);
LvSetEnum (hDev, LvDevice_GainAuto, LvGainAuto_Off);
double FloatVal;
LvGetFloat (hDev, LvDevice_Gain, &FloatVal);
```

GEN<i>CAM

```
// switch analog gain to automatic mode
write_enum (GainSelector, AnalogAll)
write_enum (GainAuto, Continuous)
// switch digital gain to automatic mode
write_enum (GainSelector, DigitalAll)
write_enum (GainAuto, Continuous)
// switch to automatic black level mode
write_enum (BlackLevelSelector, All)
write_enum (BlackLevelAuto, Continuous)
// switch the analog gain back to manual mode,
// reading the newly adjusted value
write_enum (GainSelector, AnalogAll)
write_enum (GainAuto, Off)
read_float (Gain)
```

When using automatic gain together with the automatic exposure feature (Section 6.6.2.1, “Automatic exposure” [p. 105]), the auto exposure takes precedence. Automatic gain is inactive as long as the automatic exposure functionality does not reach its limits — only then the automatic gain becomes active as well.

The automatic gain functionality might work against the manual exposure time settings. When increasing the exposure time, the automatic gain function will decrease the gain to reach again the target luminance level.

The low pass filter applied in the automatic gain algorithm can cause, that the automatic gain value never fully reaches the limits (min/max) of the manual gain settings.

6.10. Luminance and color handling

The section discusses camera features controlling the luminance and color information in the digitized image. The features include in particular the look-up table (LUT) and color transformation matrix, but the section discusses also several "higher level" functions that can be implemented using those primitives — for example gamma correction, binarization or white balance.

6.10.1. Look-up table (LUT)

The LUT (look-up table) is a trivial preprocessing technique allowing to map each input pixel value to a corresponding output pixel value. The look-up table operation is applied separately to every image pixel. The PicSight and CheckSight cameras use two types of LUTs, depending on the currently used pixel format (Section 6.5, "Image format" [p. 85]):

RGB LUT — Used only with the RGB/BGR pixel formats. The LUT consists of three tables, each controlling separately given color channel.

Luminance LUT — Used for all other pixel formats, ie. the monochrome and Bayer encoded ones. A single look-up table controls the luminance (gray value) of individual pixels.

When controlling the LUT, you first need to select the table you want adjust (luminance, red, green or blue) in the *LUT Selector* enumeration. The choice depends on the camera model and current pixel format. Having the table selected, you can loop through the entire range of *LUT Index* and adjust the *LUT Value* for each of them — we'll demonstrate that in a sample code below. Finally, the LUT operation can be switched on or off using the *LUT Enable* flag — when switched off, the data flow bypasses the LUT operation, but the LUT contents itself remains intact, so it is possible to switch it on again later, without need to refill the table.

The luminance LUT can be used to perform various operations, such as

- invert the image
- binarize the image at given threshold
- apply gamma correction
- adapt brightness and contrast
- simulate gain and black level operations
- etc.

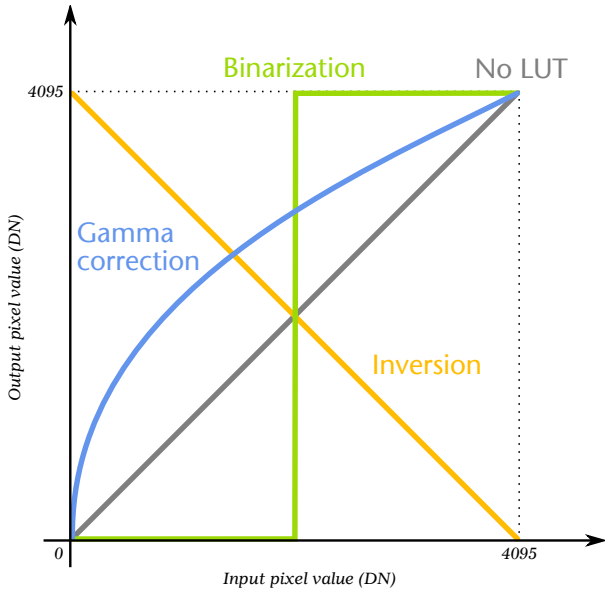


Figure 6.41. Examples of functions implemented through luminance LUT

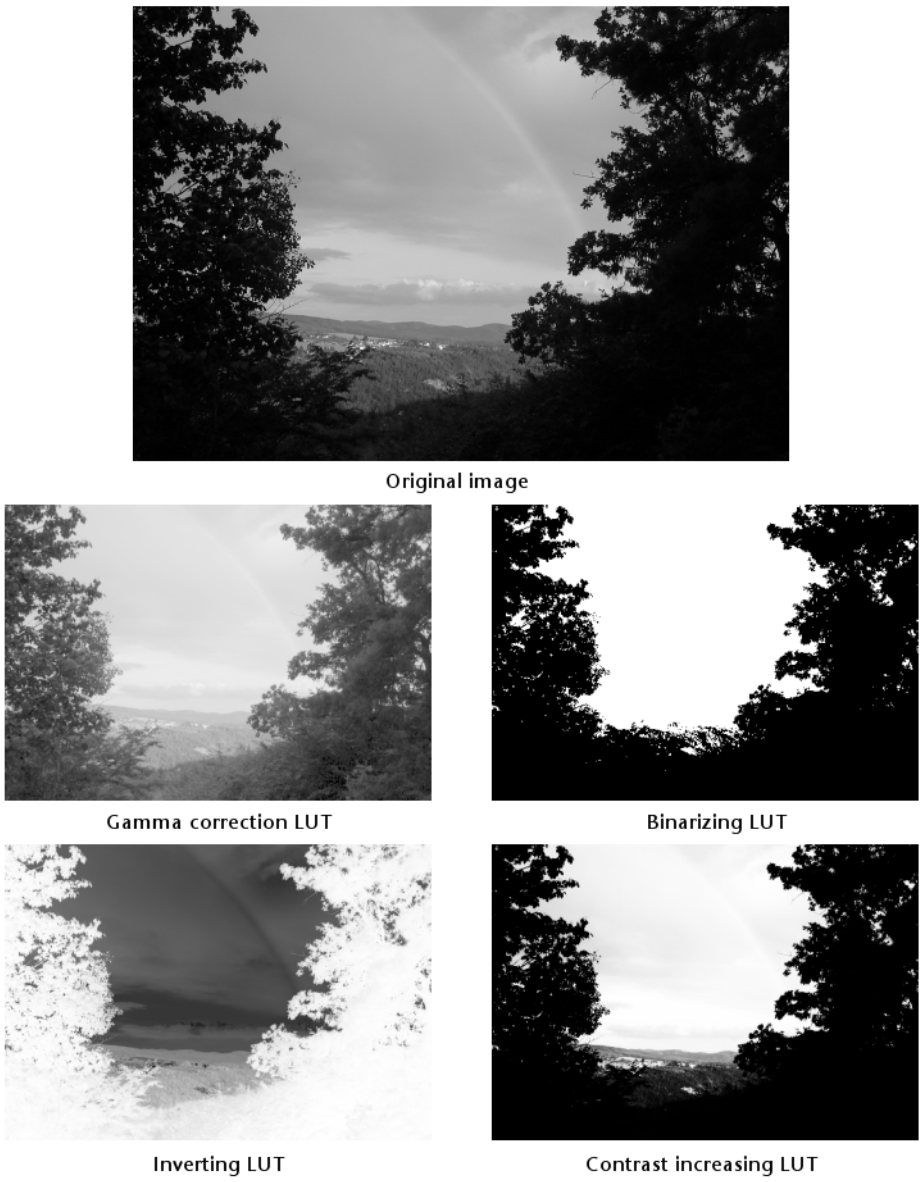


Figure 6.42. Luminiance LUT function effects illustrated

The camera internally always work with the image data read out from the sensor in 12-bit format, regardless the target format transferred to the host application. The luminance LUT thus always uses 12-bit depth at both input and output — meaning that the luminance LUT has always 4096 entries (index 0—4095) and each LUT value has also range of 0—4095, 0 corresponding with black, 4095 with white pixel.

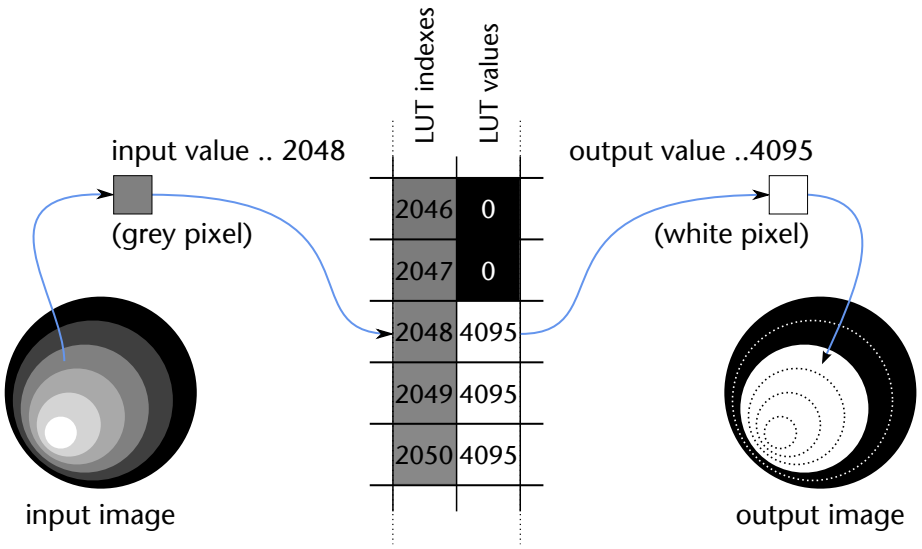


Figure 6.43. 12-bit luminance LUT used as a binarization function

Simplon

```
LvHDevice hDev; // open the device to get the handle
// adjust the luminance LUT...
LvSetEnum (hDev, LvDevice_LUTSelector, LvLUTSelector_Luminance);
// ...to invert the image
for (int i = 0; i<4095; ++i)
{
    LvSetInt32 (hDev, LvDevice_LUTIndex, i);
    LvSetInt32 (hDev, LvDevice_LUTValue, 4095-i);
}
// ...or to binarize the image at threshold 2048 (50% gray)
for (int i = 0; i<4095; ++i)
{
    LvSetInt32 (hDev, LvDevice_LUTIndex, i);
    LvSetInt32 (hDev, LvDevice_LUTValue, (i<2048) ? 0 : 4095);
}
// don't forget to enable the LUT
LvSetBool (hDev, LvDevice_LUTEnable, 1);
// disable the LUT later if needed (LUT contents will be preserved)
LvSetBool (hDev, LvDevice_LUTEnable, 0);
```

GEN<i>CAM

```
// adjust the luminance LUT...
write_enum (LUTSelector, Luminance)
// ...to invert the image
for (i = 0 .. 4095)
{
    write_int (LUTIndex, i)
    write_int (LUTValue, 4095-i)
}
// ...or to binarize the image at threshold 2048 (50% gray)
for (i = 0 .. 4095)
{
    write_int (LUTIndex, i)
```

```
    write_int (LUTValue, (i<2048) ? 0 : 4095)
}
// don't forget to enable the LUT
write_bool (LUTEnable, true)
// disable the LUT later if needed (LUT contents will be preserved)
write_bool (LUTEnable, false)
```

The RGB LUT works in principle same as the luminance LUT, it only consists from three separate tables, each controlling separately given color channel. It can accomplish the same operations as the luminance LUT — modifying all three tables in the same way. It can adapt the color information in the image — adjusting all the tables separately. A typical use case is the white balance adjustment, when individual channel responses are increased or decreased to balance the colors. Both approaches may be of course combined, eg. to perform white balance and gamma correction at the same time.

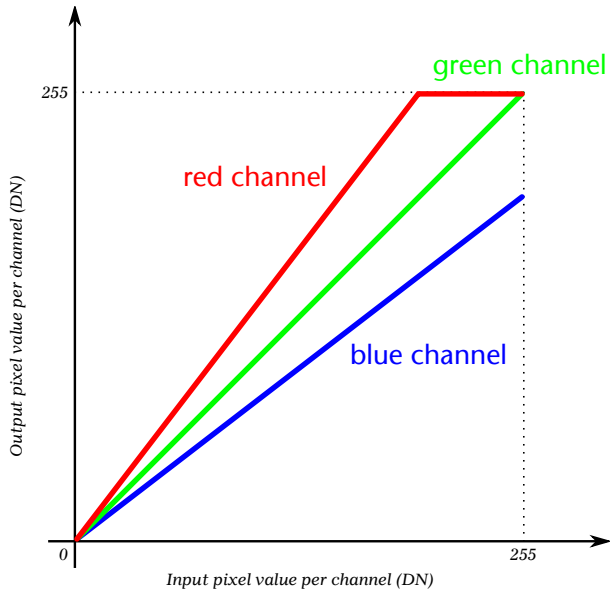


Figure 6.44. Examples of an RGB LUT used for white balancing (increased red, decreased blue channel values)

Figure 6.45. RGB LUT effect illustrated — blue channel reduced for simple white balancing

The RGB LUT is used only with true color (RGB/BGR) pixel formats, after the Bayer decoding operation, when the camera converts the image to the 3×8-bit color format. The RGB LUT thus always uses 8-bit depth at both input and output — meaning that each of the three (R-G-B) tables has always 256 entries (index 0—255) and each LUT value has also range of 0—255, 255 corresponding with fully saturated color channel of the pixel.

Note that enabling/disabling any of the three red/green/blue LUT tables actually enables/disables the entire RGB LUT. It is not possible to enable them separately.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// adjust the RGB LUT for white balancing
// increase red, decrease blue, leave green channel intact
LvSetEnum (hDev, LvDevice_LUTSelector, LvLUTSelector_Red);
for (int i = 0; i<255; ++i)
{
    LvSetInt32 (hDev, LvDevice_LUTIndex, i);
```

```
    LvSetInt32 (hDev, LvDevice_LUTValue, 1.2*i);
}
LvSetEnum (hDev, LvDevice_LUTSelector, LvLUTSelector_Blue);
for (int i = 0; i<255; ++i)
{
    LvSetInt32 (hDev, LvDevice_LUTIndex, i);
    LvSetInt32 (hDev, LvDevice_LUTValue, 0.8*i);
}
LvSetEnum (hDev, LvDevice_LUTSelector, LvLUTSelector_Green);
for (int i = 0; i<255; ++i)
{
    LvSetInt32 (hDev, LvDevice_LUTIndex, i);
    LvSetInt32 (hDev, LvDevice_LUTValue, i);
}
// don't forget to enable the LUT
// (enabling any of the three tables enables entire LUT)
LvSetBool (hDev, LvDevice_LUTEnable, 1);
// disable the LUT later if needed (LUT contents will be preserved)
LvSetBool (hDev, LvDevice_LUTEnable, 0);
```

GEN<i>CAM

```
// adjust the RGB LUT for white balancing
// increase red, decrease blue, leave green channel intact
write_enum (LUTSelector, Red)
for (i = 0 .. 255)
{
    write_int (LUTIndex, i)
    write_int (LUTValue, 1.2*i)
}
write_enum (LUTSelector, Blue)
for (i = 0 .. 255)
{
    write_int (LUTIndex, i)
    write_int (LUTValue, 0.8*i)
}
write_enum (LUTSelector, Green)
for (i = 0 .. 255)
{
    write_int (LUTIndex, i)
    write_int (LUTValue, i)
}
// don't forget to enable the LUT
// (enabling any of the three tables enables entire LUT)
write_bool (LUTEnable, true)
// disable the LUT later if needed (LUT contents will be preserved)
write_bool (LUTEnable, false)
```

6.10.1.1. LUT single access reading/writing

The per-element LUT access provides a natural interface for reading/writing the LUT, the application does not need to make any assumptions about the internal layout of the LUT. On the other hand, reading/wriging the entire LUT this way can generate significant overhead due to excessive I/O operations. In some situations, it might be desirable to eliminate this overhead, reading or writing the entire LUT in a single access.

The single-step access is performed through feature *LUT Value All*, which is based on GenICam type `IRegister`, behaving as a raw block of memory. All the rules about selecting the LUT to be accessed (*LUT Selector*) or enabling the LUT (*LUT Enable*) apply same as in case of the per-element access method, the feature *LUT Value All* just allows to replace the *LUT Index* and *LUT Value* calls with a single input/output request.

Since the *LUT Value All* feature accesses the LUT in a single step as a block of memory, it is necessary to know the internal layout of the LUT:

- Regardless of the LUT type, the size of the individual elements, as they are stored in the memory, are always 4 bytes. Naturally, not all bits of the 4-byte slot are actually used, the luminance LUT uses 12-bit, the RGB LUT's 8-bit elements. When writing the LUT as a memory block, the application is responsible to not overflow these limits.
- Beware, due to differences between the actual technologies, the endianess of the individual elements vary among Leutron Vision camera families. PicSight-GigE cameras store the LUT elements in big endian, CheckSight cameras in little endian format. The application is responsible to provide the memory block in correct format when writing. Note that these considerations apply only to the “entire memory block” access, when accessing the LUT per-element the LUT internals are shielded from the user.
- The number of elements in the LUT depends on the LUT type. As described above, the luminance LUT has always 4096 elements, while the individual R/G/B LUT's have 256 elements each.
- The entire size of the LUT in bytes is naturally product of the single element size (4B) and number of entries.
- The order of elements in the LUT is defined by their indexes. The element with index `0` is always stored at the lowest address (aligned with the start of the LUT memory block). The element with index `1` can be accessed at a 4-byte offset from the LUT beginning, etc.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// adjust the luminance LUT...
LvSetEnum (hDev, LvDevice_LUTSelector, LvLUTSelector_Luminance);
uint8_t LUTData[16384]; // 4096 * 4B
// prepare the memory block LUTData, using size and layout
// compatible with camera's internal representation of the LUT
LvSetBuffer(hDev, LvDevice_LUTValueAll, LUTData, 16384);
// don't forget to enable the LUT
LvSetBool (hDev, LvDevice_LUTEnable, 1);
```

GEN<i>CAM

```
// adjust the luminance LUT...
write_enum (LUTSelector, Luminance)
// prepare the memory block LUTData, using size and layout
// compatible with camera's internal representation of the LUT
write_raw_reg (LUTValueAll, LUTData, 16384)
// don't forget to enable the LUT
write_bool (LUTEnable, true)
```

6.10.1.2. Direct and indirect LUT control (LUT modes)

The LUT(s) can be configured either directly by adjusting the look-up table entries — or, if given camera supports it, through higher level functions built on top of the LUT functionality. An example of the higher level function is white balance, which is implemented on top of the RGB LUT.

The higher level function(s) fully control the LUT, in such case it's no more possible to modify the LUT entries directly. Therefore the user has to select the desired mode of operation. If a given camera supports various LUT operation modes, they are controlled through enumeration feature *LUT Mode*.

The basic, always available, option is *Direct*. In this case the LUT entries are controlled directly by the application. The higher level functions are not available.

Alternatively, it's possible to select a higher level function (e.g. *White Balance*). In that case, the features controlling given higher level function become available and take over the LUT control. LUT itself is still available, but in read-only mode. Individual higher level functions are discussed in separate sections.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// select the direct LUT control mode...
LvSetEnum (hDev, LvDevice_LvLUTMode, LvLUTMode_Direct);
// ...access the LUT entries directly now
// select a higher level function, here white balance
LvSetEnum (hDev, LvDevice_LvLUTMode, LvLUTMode_BalanceWhite);
// ...now LUT itself is read-only, controlled indirectly
```

GEN<i>CAM

```
// select the direct LUT control mode...
write_enum (LvLUTMode, Direct)
// ...access the LUT entries directly now
// select a higher level function, here white balance
write_enum (LvLUTMode, BalanceWhite)
// ...now LUT itself is read-only, controlled indirectly
```

6.10.1.3. White balance

White balance function is adjustment of intensities of individual color (R-G-B) channels to compensate for the unwanted color shift, e.g. because of non-neutral ambient illumination of the scene. The white balancing is controlled through amplification factors to be applied on the individual R-G-B color channels. Note that white balance makes only sense when using a true RGB pixel format.

The camera implements the white balance functionality using the RGB LUT — the white balance factors (“ratios”) are applied as linear functions in the individual R-G-B LUT's. To make the white balance functinality avilable (if supported by the camera), first switch *LUT Mode* to *White Balance*.

To apply the desired white balance factors, select progressively the individual color channels (*Red*, *Green*, *Blue*) in the *Balance Ratio Selector* enumeration and for each channel configure the desired amplification factor through *Balance Ratio*. The *Balance Ratio* is a floating point factor allowing to amplify the given color channel within certain limits.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// select the "white balance" LUT mode
LvSetEnum (hDev, LvDevice_LvLUTMode, LvLUTMode_BalanceWhite);
// adjust white balance:
// increase red, decrease blue, leave green channel intact
LvSetEnum (hDev, LvDevice_BalanceRatioSelector,
                 LvBalanceRatioSelector_Red);
LvSetFloat (hDev, LvDevice_BalanceRatio, 1.2);
LvSetEnum (hDev, LvDevice_BalanceRatioSelector,
                 LvBalanceRatioSelector_Green);
LvSetFloat (hDev, LvDevice_BalanceRatio, 1.0);
LvSetEnum (hDev, LvDevice_BalanceRatioSelector,
                 LvBalanceRatioSelector_Blue);
LvSetFloat (hDev, LvDevice_BalanceRatio, 0.8);
```

GEN<i>CAM

```
select the "white balance" LUT mode
write_enum (LvLUTMode, BalanceWhite)
write_enum (BalanceRatioSelector, Red)
write_float (BalanceRatio, 1.2)
write_enum (BalanceRatioSelector, Green)
write_float (BalanceRatio, 1.0)
write_enum (BalanceRatioSelector, Blue)
write_float (BalanceRatio, 0.8)
```

Besides the “manual” setting of the white balancing ratios, the camera might compute them automatically based on an actually acquired image. To instruct the camera to compute the new ratios, switch the *Automatic White Balance* feature to *Once* (the default value of this enumeration is *Off*). As soon as switched to *Once*, the camera will automatically evaluate the acquired image and compute new white balancing factor — considering, that the acquired image is color neutral, i.e. suitable for white balance calibration. When computed, the *Automatic White Balance* will switch back to *Off* and the newly computed values will be available through *Balance Ratio Selector*/*Balance Ratio*. Of course, the new values will immediately become effective. When required, the process should be repeated again.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// select the "white balance" LUT mode
LvSetEnum (hDev, LvDevice_LvLUTMode, LvLUTMode_BalanceWhite);
// trigger computation of new white balance factors
LvSetEnum (hDev, LvDevice_BalanceWhiteAuto,
                 LvBalanceWhiteAuto_Once);
// when finished, it switches back to LvBalanceWhiteAuto_Off
// and new balance ratios are immediately applied
```

GEN<i>CAM

```
select the "white balance" LUT mode
write_enum (LvLUTMode, BalanceWhite)
// trigger computation of new white balance factors
write_enum (BalanceWhiteAuto, Once)
// when finished, it switches back to LvBalanceWhiteAuto_Off
// and new balance ratios are immediately applied
```

Note, that computation of the white balancing factors based on the acquired image can be done even if the output pixelf format is not true RGB, but a Bayer encoded format. In such case the camera does not apply them directly (no RGB LUT with Bayer encoded pixel format), it just provides the recommended values (read-only) as a hint for the application, which can apply them after decoding the Bayer encoded image.

6.10.2. Color transformation

The color transformation functionality allows to adjust the color information through applying a matrix operation on every single pixel. The operation is applicable only for true color RGB/BGR output pixel formats (Section 6.5, “Image format” [p. 85]) since it takes the R-G-B channel values of each pixel as input.

The color transformation functionality can be used to perform various operations, such as

- image color improvement/correction, advanced white balancing, channel overlap elimination
- color to grayscale conversion
- format conversion (eg. RGB to YUV)
- adjusting color saturation
- adjusting brightness and contrast of color images
- hue rotation
- etc.

The color transformation functionality might be only available on specific camera models. Please contact Leutron Vision to check availability of the feature and its detailed documentation for given model.

Figure 6.46. Artificial example illustrating the color transformation (in this case full color space rotation)

6.10.3. Gamma correction, brightness and contrast

Simplon API provides several utility functions helping to implement some common image conditioning operations using the camera's LUT functionality. It can, for example, configure the LUT to apply desired gamma correction, optionally combined with brightness and contrast adjustment. Simplon API would automatically find out the currently active LUT type (luminance or RGB LUT), compute the LUT values based on the required inputs and adjust the LUT. The details are documented in Simplon programmer's guide.

6.10.4. White balance

The color interpretation in the acquired image might not always exactly correspond with the reality, for example because of non-neutral ambient illumination in the scene — some color components might be emphasized more than others. The goal of the white balancing operation

is to adjust the image colors so that color neutral objects (white or gray) are represented as such in the acquired image data. This operation can be achieved through

- RGB LUT — linearly amplifying the color channels that are too weak in the image and vice versa
- Color transformation matrix — which can also simply amplify or curtail individual channels, but can also "shift" the response of one channel to another one

Simplon API provides functions to help with white balancing. It would query the color handling functions supported by the connected camera (RGB LUT or color transformation matrix), inspect the target image buffer supplied by the application, calculate the necessary parameters needed to white-balance that buffer and configure the camera with those parameters. White balancing can be combined with the other LUT based operations, such as gamma correction, described above. The details are documented in Simplon programmer's guide.

6.11. Chunk data

This section discusses the additional info fields (so called "chunk data") optionally appended to every acquired image.

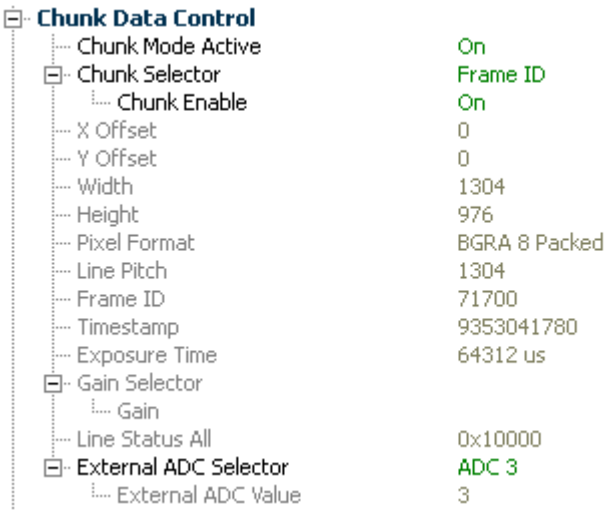


Figure 6.47. Example GUI showing the chunk data features

The chunk data are additional data optionally appended to every acquired image. The data can be retrieved through the feature tree, similarly as regular camera features. The individual chunks contain information about the camera status and configuration of certain important features at the time the image was acquired, thus providing additional information about the image. In triggered mode, the chunk data are collected in the moment of applying the frame trigger (effective exposure start). In free running mode, the chunk data are collected upon start of sensor readout.

The application can selectively choose, which chunks should be delivered. The chunk data are always delivered in the same buffer with the actual image data. The status of chunk data configuration thus affects the "payload size" (Section 6.4.1, "Acquired image data size" [p. 84]) and the application must consider that when allocating the acquisition buffers. The chunk data are always appended to the *end* of the buffer (in a standardized format), the image data are always in the beginning. Accessing the image data itself is thus not affected by the fact, whether the chunk data are used or not.

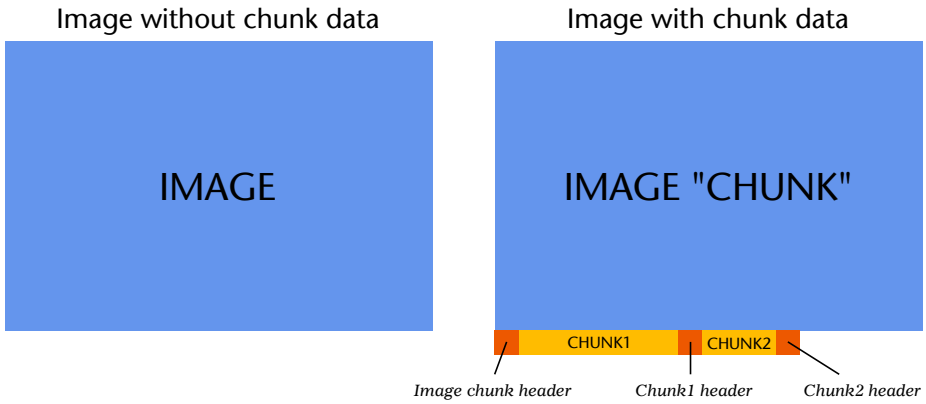


Figure 6.48. Chunk data layout, the image data are always at the "start" of the buffer, chunk data parsed from the end of the delivered buffer

Naturally, since the chunk data increase the image buffer size, the bandwidth required to deliver the image stream to the application also increases. However, the chunk data size is usually very small compared to the image data size, therefore its influence is marginal.

The GigE Vision compatible cameras (PicSight-GigE and PicSight-Smart) naturally deliver the chunk data with images using the GigE Vision payload types intended for chunk data. Other camera families, in particular CheckSight/CheckSight-II, deliver the image with chunk data through GenTL image acquisition engine. Note that all camera families, including CheckSight/CheckSight-II, deliver the chunk data in GigE Vision compatible format. Therefore every library/application capable of parsing the GigE Vision chunk data can use the same mechanism to parse also CheckSight/CheckSight-II chunk data.

Not all GigE Vision and GenICam based software packages are capable of working with the chunk data. If accessing the cameras through other library than Simplon API, please check its documentation to see if it supports chunk data parsing and delivery. Note that for GigE Vision cameras (PicSight-GigE and PicSight-Smart), part of the information available through the chunk data are also delivered in GigE Vision control packets (data leader and trailer). If your library provides access to those fields, they can be used alternatively instead of the chunk data.

In Simplon API, the library's acquisition engine delivers acquired image buffers. In chunk mode, it is responsibility of the application to specify, when the chunk data of a given buffer should be parsed and "attached" to the camera's feature tree. As soon as the library performs the chunk data processing in the given buffer, the values of individual chunks are available through corresponding camera features. The values will be available until another buffer's chunk data are processed — then the features are updated with new values. Note that it is important to process and read the chunk data before returning the buffer to the acquisition engine ("queuing" the buffer) — after that the buffers is owned by the acquisition engine and its contents is undefined. All needed details are documented in Simplon programmer's guide

In a 3rd party library, the mechanism might be similar, but it is important to study its documentation carefully to be sure how to retrieve the chunk data reliably.

6.11.1. Enabling chunk data

By default, the chunk data delivery is switched off. The application should switch on each particular chunk type it's interested in. It does so by selecting the desired chunk type in *Chunk Selector* feature and set the *Chunk Enable* boolean flag. It should repeat this procedure for all chunk types it wishes to monitor. Finally, the application must enable the chunk delivery mode through the *Chunk Mode Active* boolean feature. This feature allows to globally switch the chunk mode on and off, independently on which chunk types are currently selected.

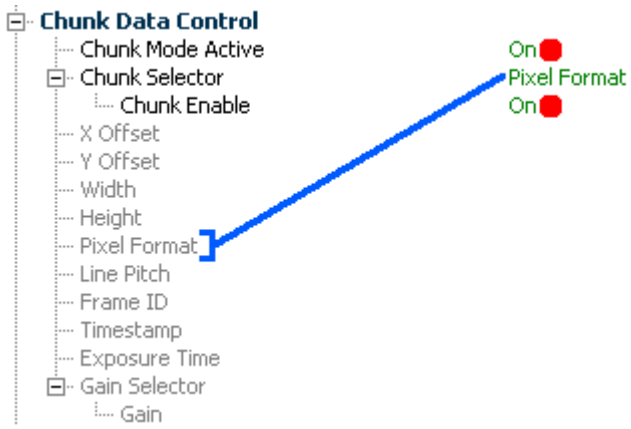


Figure 6.49. Chunk “Pixel Format” enabled, associated feature highlighted

Simplon

```
LvHDevice hDev; // open the device to get the handle
// switch to the chunk mode
LvSetBool (hDev, LvDevice_ChunkModeActive, 1);
// and enable the "ExposureTime" chunk delivery
LvSetEnum (hDev, LvDevice_ChunkSelector, LvChunkSelector_ExposureTime);
LvSetBool (hDev, LvDevice_ChunkEnable, 1);
// now the camera appends the exposure time chunk
// to every acquired image
// the application can process the chunk data in each delivered buffer
// and read associated feature values
while (NewAcquiredBuffersComing)
{
    LvHBuffer hBuf; // obtain handle of the acquired buffer
    LvBufferParseChunkData (hBuffer, 0);
    double FloatVal;
    LvGetFloat (hDev, LvDevice_ChunkExposureTime, &FloatVal);
}
// switch off the chunk mode, if desired
LvSetBool (hDev, LvDevice_ChunkModeActive, 1);
```

GEN<i>CAM

```
// switch to the chunk mode
write_bool (ChunkModeActive, true)
// and enable the "ExposureTime" chunk delivery
write_enum (ChunkSelector, ExposureTime)
write_bool (ChunkEnable, true)
// now the camera appends the exposure time chunk
// to every acquired image
// the application can process the chunk data in each delivered buffer
// and read associated feature values
while (NewAcquiredBuffersComing)
{
    // before reading the values, pass the buffer for chunk data parsing
    // unless your library does that implicitly
    read_float (ChunkExposureTime)
}
```

```
// switch off the chunk mode, if desired
write_bool (ChunkModeActive, false)
```

In Simplon API to read the chunk data values delivered in a buffer, the application has to pass the acquired buffer's handle to the chunk data parsing function, `LvBufferParseChunkData()`. Details including sample code are listed in Simplon programmer's guide. In other software packages, the procedure might be quite similar.

The Simplon Explorer application will read the chunk data automatically, when enabled. It will parse the chunk data in every acquired buffer and update corresponding values in the feature tree. What it displays in the feature tree in a moment is of course the value associated with the currently displayed image.

6.11.2. Chunk data features naming convention

As discussed above, each supported chunk is listed in the *Chunk Selector* enumeration and can be selectively enabled or disabled. The value of each supported and enabled chunk can be read through corresponding read only feature: for a chunk with name *XXX*, the chunk value feature is named `Chunk`*XXX*. When referring to an "array" of values of the same type (such as analog and digital `Gain`), the values are indexed using a selector feature, as usual (eg. `ChunkGainSelector`, `ChunkGain`).

For chunks carrying values of a camera configuration feature (such as `ExposureTime`), the chunk name is equivalent as the feature name itself. Of course, data types and value ranges are specific to each chunk.

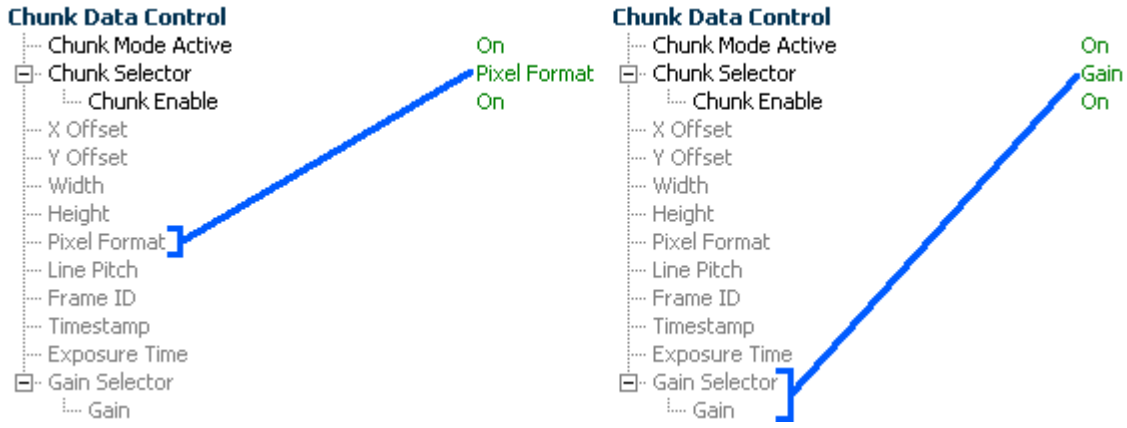


Figure 6.50. Chunk data feature naming illustrated

6.11.3. Chunk types

The Leutron Vision cameras currently support following chunk types:

Width (`Width`)	Width of the image in the buffer.
Height (`Height`)	Height of the image in the buffer.
X Offset (`OffsetX`)	Horizontal AOI offset used when acquiring the image in the buffer.
Y Offset (`OffsetY`)	Vertical AOI offset used when acquiring the image in the buffer.
Pixel Format (`PixelFormat`)	Pixel format of the image in the buffer.
Line Pitch (`LinePitch`)	Line pitch (in bytes) of the image in the buffer.

Timestamp (`Timestamp`)	Value of the camera's timestamp counter at the moment the image was acquired (64-bit integer). The camera timestamp frequency is 1MHz, ie. the ticks correspond with time unit of 1 µs.
Frame ID (`FrameID`)	Value of the frame counter associated with the image in the buffer (64-bit integer).
Exposure Time (`ExposureTime`)	Exposure time used to acquire the image in the buffer.
Gain (`Gain`)	Gain used to acquire the image in the buffer. Since some camera models support two types of gain, analog and digital (refer to Section 6.9, "Analog controls" [p. 127]), the `ChunkGain` feature comes with a selector, `ChunkGainSelector`. To read the analog gain chunk (if supported by the camera), set the selector to `AnalogAll`, to read the digital gain, set `DigitalAll`.
Black Level (`BlackLevel`)	Black level used to acquire the image in the buffer.
Line Status All (`LineStatusAll`)	Bitfield reflecting status of all digital I/O lines (Section 6.7, "Digital I/O, counters and timers" [p. 113]) at the time the image was acquired. The least significant bit corresponds with `Line1`, next bit with `Line2` etc.
External ADC Value (`LvExternalADCValue`)	Status of the external ADC inputs (Section 6.14, "External device control" [p. 160]) at the time the image was acquired. Since the cameras are equipped with multiple external ADC's, the `ChunkLvExternalADCValue` feature comes with a selector, `ChunkLvExternalADCSelector`. To read a particular external ADC value chunk (if supported by the camera), select it first in the selector.
Trigger Delayed (`LvTriggerDelayed`)	Flag indicating, whether the frame trigger for the current image was delayed, because it was issued too early (before the camera could accept it). Note that the way how early triggers are processed (cache/drop) can be controlled through feature *Trigger Caching* (Section 6.6, "Acquisition and trigger control" [p. 101]). Dropped triggers are signalled through asynchronous events (Section 6.12, "Asynchronous events" [p. 151]).
Strobe Dropped (`LvStrobeDropped`)	Flag indicating (for cameras supporting the integrated strobe ring only), whether the strobe pulse was dropped for the current image due to hardware constraints — in particular if the timing did not match the device capabilities (the trigger arrived before the device could accept it). Note that the way how early strobe requests are processed (cache/drop) can be controlled through feature *Strobe Drop Mode*.
Smart Application String (`LvSmartAppString`)	A string that might be appended to the image by the "smart application" (running on PicSight-Smart). The contents and length of the string is defined by the respective application.
Smart Application Signed Integer (`LvSmartAppInt`)	One or more signed integer values that might be appended to the image by the "smart application" (running on PicSight-Smart). The contents and number of the values is defined by

the respective application. Since multiple values might be delivered, the `ChunkLvSmartAppInt` feature comes with a selector, `ChunkLvSmartAppIntSelector`, which is an integer with range from 1 to the number of values in the chunk. To read the desired value, set the selector to point to the value of question.

Smart Application Unsigned Integer (`LvSmartAppUint`)

One or more unsigned integer values that might be appended to the image by the "smart application" (running on PicSight-Smart). The contents and number of the values is defined by the respective application. Since multiple values might be delivered, the `ChunkLvSmartAppUint` feature comes with a selector, `ChunkLvSmartAppUintSelector`, which is an integer with range from 1 to the number of values in the chunk. To read the desired value, set the selector to point to the value of question.

Smart Application Register (`LvSmartAppRegister`)

A raw register (memory buffer) that might be appended to the image by the "smart application" (running on PicSight-Smart). The contents and length of the register is defined by the respective application.

6.12. Asynchronous events

This section discusses the asynchronous event (and associated data) delivery.

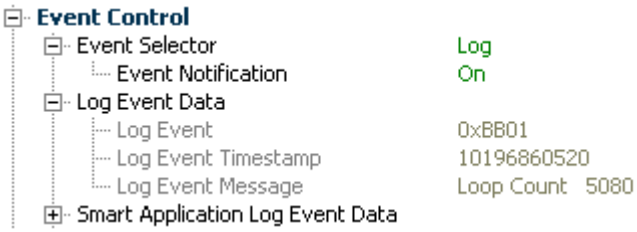


Figure 6.51. Example GUI showing the asynchronous event features

Besides the acquired images, and possibly additional "chunk" data appended to images(Section 6.11, "Chunk data" [p. 146], the cameras can deliver additional "asynchronous" messages. The events are delivered through a separate channel, independent on the streaming channel (images) and control channel (camera configuration), therefore "asynchronous".

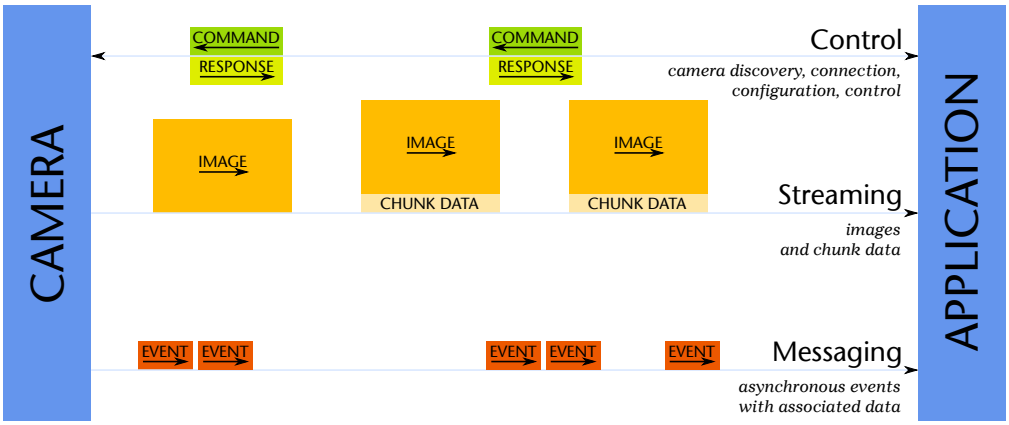


Figure 6.52. Camera communication channels

The events can be just a simple messages (such as signalling occurrence of an incident, such as error), but usually they carry additional data. The data can be retrieved through the feature tree, similarly as regular camera features. All the events have associated timestamp, which is the value of the camera's timestamp counter (refer to Section 6.3, “Information about the camera” [p. 80]) at the time the event was generated. The camera timestamp frequency is 1MHz, ie. the ticks correspond with time unit of 1 µs.

The GigE Vision compatible cameras (PicSight-GigE and PicSight-Smart) naturally deliver the events through the GigE Vision message channel. Other camera families, in particular CheckSight/CheckSight-II, deliver the events through GenTL event signalling mechanism. Note that all camera families, including CheckSight, deliver the event data in GigE Vision compatible format. Therefore every library/application capable of parsing the GigE Vision event data can use the same mechanism to parse also CheckSight/CheckSight-II event data.

Not all GigE Vision and GenICam based software packages are capable of working with the asynchronous events. If accessing the cameras through other library than Simplon API, please check its documentation to see if it supports asynchronous event delivery and event data parsing.

In Simplon API, the library deliver callbacks to your application, whenever an asynchronous event arrives (provided that you have registered for the callback delivery). The callbacks will signal all the features corresponding with the data delivered with the event — announcing the application that it can read the new event data. It is important to read the data directly withing the callback, which is executed in the context of the Simplon API event delivery thread. After leaving the callback, the event delivery thread continues and the event data feature values might eventually be overwritten by a following event. All needed details are documented in Simplon programmer's guide.

In a 3rd party library, the mechanism might be similar, but it is important to study its documentation carefully to be sure how to retrieve the event data reliably.

6.12.1. Enabling events

By default, the event delivery is switched off. The application should switch on each particular event type it's interested in. It does so by selecting the desired event type in *Event Selector* feature and adjust the *Event Notification* feature to *On* (note that due to the compatibility with the SFNC standard, this feature is not a boolean, but an enumeration with two options, *On* and *Off*). It should repeat this procedure for all event types it wishes to monitor.

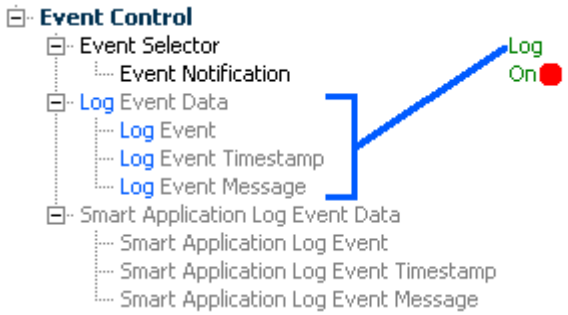


Figure 6.53. Event type “Log” enabled, subcategory with features carrying the associated data highlighted

```
LvHDevice hDev; // open the device to get the handle
// enable the "Log" events delivery
LvSetEnum (hDev, LvDevice_EventSelector, LvEventSelector_LvLog);
LvSetEnum (hDev, LvDevice_EventNotification, LvEventNotification_On);
```

```
// now the camera starts generating the log events whenever appropriate
// provided that a callback was registered for the log event (LvDevice_EventLvLo
// the application can monitor them and read associated event data
while (NewLogEventsComing)
{
    char StringVal[1024];
    LvGetString (hDev, LvDevice_EventLvLogMessage, StringVal, 1024);
    int64_t Int64Val;
    LvGetInt64 (hDev, LvDevice_EventLvLogTimestamp, &Int64Val);
}
// disable the "Log" events delivery
LvSetEnum (hDev, LvDevice_EventSelector, LvEventSelector_LvLog);
LvSetEnum (hDev, LvDevice_EventNotification, LvEventNotification_Off);
```

GEN<i>CAM

```
// enable the "Log" events delivery
write_enum (EventSelector, LvLog)
write_enum (EventNotification, On)
// now the camera starts generating the log events whenever appropriate
// provided that event delivery was registered for the log event (EventLvLog)
// the application can monitor them and read associated event data
while (NewLogEventsComing)
{
    // before reading the values, pass the event buffer for data parsing
    // unless your library does that implicitly
    read_string (EventLvLogMessage)
    read_int (EventLvLogTimestamp)
}
// disable the "Log" events delivery
write_enum (EventSelector, LvLog)
write_enum (EventNotification, Off)
```

In Simplon API there are two actions needed to receive the asynchronous event notifications. First, the event receiving and dispatching thread must be started (`LvEventOpen(LvEventType_FeatureDevEvent)`, `LvEventStartThread()`). Second, callbacks need to be registered on the features associated with the desired event (`LvRegisterFeatureCallback()`) — the rules about event feature naming are discussed below in Section 6.12.2, “Event features naming convention” [p. 154]. Details including sample code are listed in Simplon programmer's guide. In other software packages, the procedure might be quite similar.

The Simplon Explorer application will capture the events automatically, when enabled. It will parse the event data and update corresponding values in the feature tree. What it displays in the feature tree in a moment is of course the value associated with the last delivered event. The event history is not recorded in Simplon Explorer.

The only exception are the logging messages, which might be recorded and displayed in a separate panel. This functionality is, however, only intended for customer support. If the support case requires it, our support staff will provide instructions how to capture the relevant camera log. Complete user interface of the Simplon Explorer is described in Simplon programmer's guide.

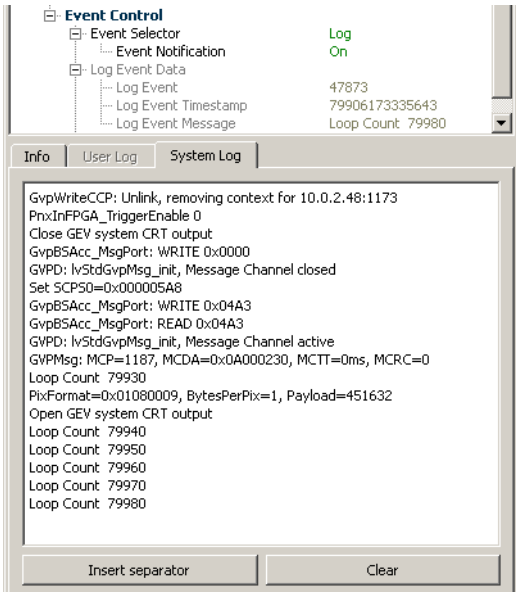


Figure 6.54. Camera log displayed in Simplon Explorer

6.12.2. Event features naming convention

The naming of event related features follows the recommendation of the GenICam SFNC standard. As described earlier, each supported event is listed in the *Event Selector* enumeration and can be selectively enabled or disabled. Each supported event comes with a set of additional features following fixed naming scheme.

Let's describe the naming scheme for a hypothetical event with name *XXX*:

`Event`*XXX*`Data` — A subcategory of the `EventControl` category. This subcategory contains all the features specific to the *XXX* event.

`Event`*XXX* — Read only integer feature. Its main purpose is that it can be used for registering the callback, when listening for given events. Each time the event of this type is generated by the camera, the callback would be called (note that other libraries than Simplon API might use other mechanism than callbacks for signalling the event). Note that when monitoring an event carrying some event data, registering the callback on the data field would also work. The `Event`*XXX* feature is just available for every single event type.

The value of the `Event`*XXX* integer is always the ID of the respective event type. Note that every event type is internally identified by a unique numeric ID. The ID is, however, not important for most applications, and especially to Simplon API based ones, since it is not needed for event monitoring.

`Event`*XXX*`Timestamp` — Every single generated event carries a timestamp — value of the camera's timestamp counter at the time the event was generated. It is a 64-bit integer value.

`Event`*XXXyyy* — Feature(s) carrying value(s) of the event data "*yyy*" associated with the given event. For each event type there might be multiple event data fields, single field or even none. The data type and value range of individual fields might also be different for each event type.

The same features are used for all instances of given event, so at every moment, they provide the value(s) carried by the last delivered event. As soon as the next event instance is delivered, the values are overwritten by the new data.

When using a software package that does not support event delivery, no callbacks will be delivered and it will not be possible to read any event data. The event related features will not be even available for reading.

To demonstrate the rules on a real example — the "log" events of PicSight-GigE cameras are identified by `LvLog` entry in the `EventSelector`. The name of the subcategory related to the event data is `EventLvLogData`. The primary target for registering the callback for this event is `EventLvLog`. The subcategory also contains the "timestamp" feature `EventLvLogTimestamp` and a single data feature specific to this event type, in this case `EventLvLogMessage`. Note that other event types might carry multiple event specific data fields as well as none of them.

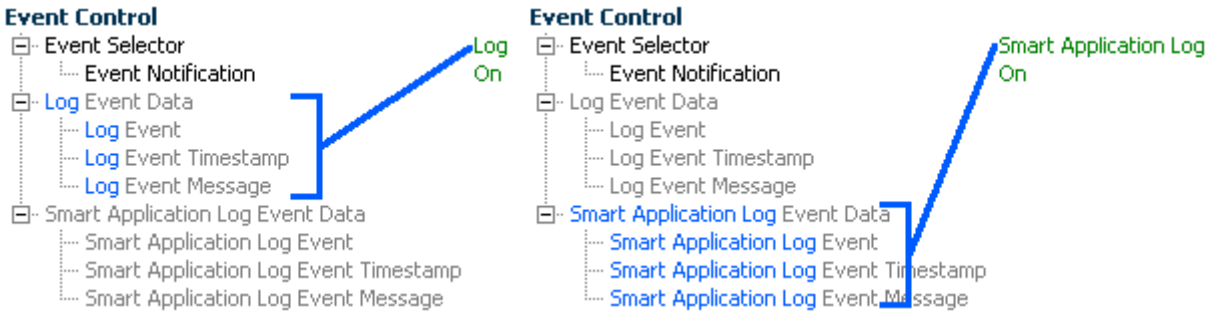


Figure 6.55. Event naming convention illustrated

6.12.3. Event types

The Leutron Vision cameras currently support following event types:

Log (`LvLog`)

Camera logging messages. Intended for troubleshooting and support purposes. If needed, our support staff will provide instructions, how to capture the entire log. Please note that the log contents is intended to be interpreted solely by our support staff, it does not provide any information useful for the camera user. Therefore please do not try to interpret its contents or report to us log contents that you potentially find "suspiciouis". Under normal circumstances the log events should be switched off so that they do not occupy the camera link.

Event ID for this event type is `0xBB01`.

Event specific data fields:

- `EventLvLogMessage` — log message as string of variable length. If a log message is too long (longer than the maximal possible event data size allowed by the GigE Vision standard), the camera splits it among multiple events.

Trigger Dropped (`LvTriggerDropped`)

Message signalling that a frame trigger was dropped, because it was issued too early (before the camera could accept it).

Note that the way how early triggers are processed (cache/drop) can be controlled through feature *Trigger Caching* (Section 6.6, "Acquisition and trigger control" [p. 101]). Info

about delayed triggers are delivered through within chunk data for the corresponding image (Section 6.11, “Chunk data” [p. 146]).

Event ID for this event type is `0xBB07`.

Event specific data fields: none, this event type serves as a pure “signal” carrying just the timestamp.

Smart Application Log (`LvSmartAppLog`)

The smart application log event is analogical to the regular log event, but it's dedicated for use by smart applications (running on PicSight-Smart cameras). The contents and purpose of the log messages is thus strictly specific to the running smart application.

Event ID for this event type is `0xBB02`.

Event specific data fields:
- `EventLvSmartAppLogMessage` — smart application “log” message as string of variable length. If a message is too long (longer than the maximal possible event data size allowed by the GigE Vision standard), the camera splits it among multiple events.

Smart Application String (`LvSmartAppString`)

An asynchronous string message generated by the “smart application” (running on PicSight-Smart). The contents and length of the string is defined by the application. Strings delivered through this event type are not considered by the logging facilities.

Event ID for this event type is `0xBB03`.

Event specific data fields:
- `EventLvSmartAppStringValue` — the string itself.

Smart Application Signed Integer (`LvSmartAppInt`)

Asynchronous message carrying one or more signed integer values, generated by the “smart application” (running on PicSight-Smart). The contents and number of the values is defined by the application.

Event ID for this event type is `0xBB04`.

Event specific data fields:
- `EventLvSmartAppIntSelector` — selects one of the values delivered in the message. It is an integer with range from 1 to the number of values in the message. To read the desired value, set the selector to point to the value of question.
- `EventLvSmartAppIntValue` — reads the signed integer value the selector points to.

Smart Application Unsigned Integer (`LvSmartAppUint`)

Asynchronous message carrying one or more unsigned integer values, generated by the “smart application” (running on PicSight-Smart). The contents and number of the values is defined by the application.

Event ID for this event type is `0xBB05`.

Event specific data fields:

- `EventLvSmartAppUintSelector` — selects one of the values delivered in the message. It is an integer with range from 1 to the number of values in the message. To read the desired value, set the selector to point to the value of question.
- `EventLvSmartAppUintValue` — reads the unsigned integer value the selector points to.

Smart Application Register (`LvSmartAppRegister`)

A raw register (memory buffer) message generated by the "smart application" (running on PicSight-Smart). The contents and length of the register is defined by the application.

Event ID for this event type is `0xBB06`.

Event specific data fields:
- `EventLvSmartAppRegisterValue` — the "register" feature carrying the raw memory data.

6.13. User sets (saving/loading camera configuration)

This section discusses options to store and reuse camera configuration sets.

Figure 6.56. Example GUI showing the user sets' features

6.13.1. Features defining camera configuration

The set of features that the user can control make up the camera configuration. The complete current configuration can be at any point stored away and later re-loaded to bring the camera back to the status when the configuration snapshot was saved. The configuration user set handling can be performed directly by the camera or by the application itself — both approaches are discussed below.

It's important to understand, which features belong to the user set and will thus be "remembered". Let's list few rules:

- Refer to Section 6.16, "Feature summary" [p. 167] — features marked as "streamable" will be stored in the user sets.
- In general, the user sets consist of all the read-writable features, ie. those that can be written and retrieved back.
- Features depending on a "selector", ie. array-like features are stored for all possible selector values (the entire array is always stored and re-loaded). If a feature is available and read-writable only for some selector values, than only those entries will be part of the user set.
- Naturally read-only features (such as serial number or model name) are not part of the user sets. Remember, that being read-only does not necessarily mean the feature is constant. For example, if in DHCP mode, the camera might have different IP address at each run, but still the "current IP address" feature will not be part of the user set — ie. the IP address would not be restored.
- Commands are not part of the user set. No commands can be executed while re-loading the configuration.

- There are few exceptions among the read-writable features, that do not belong to the user sets. These are the low-level features used to set-up the basic connection with the camera. These features are always under direct access of the camera controlling library and trying to restore their values among sessions makes no sense.
- The same applies for features defining identify of the camera (such as its persistent IP address or user name). These values are anyway persisted by the camera — and trying to apply them to multiple cameras through software controlled configuration set functionality would lead to unwanted results.
- The default user configuration set used during camera boot is also not stored.

The user sets are designed to be expectable, storing a configuration away and reloading it back brings the camera to the same status as when the configuration snapshot was made.

The user set storing/loading should be performed only when acquisition is not active. Once the acquisition is started, some features are not writable any more and the camera will not allow adjusting their values or load a user set.

6.13.2. Camera's user configuration sets

The camera can store up to five configuration sets, one with factory default settings (*Default User Set*) and four user configurable sets (*User Set 1* to *User Set 4*).

The set with factory defaults is fixed, cannot be written. The settings include free running working mode and full resolution image. Also defaults for other settings are chosen to be expectable — if needed, the default behavior for each feature is documented together with the feature itself.

The user sets can be used to save “snapshots” of particular camera configurations and reuse them later. After adjusting the desired configuration, it can be stored to the camera's persistent memory by selecting the desired set in *User Configuration Set Selector* feature and invoking the *User Set Save* command. To load them later, one invokes the *User Set Load* command, having the desired set selected.

Note that after loading a configuration set, virtually any feature might change its value. Therefore the application might need to re-read the entire feature tree to know the new camera configuration.

Finally, it is possible to choose, which of the configuration sets will be applied during camera boot. Usually the factory defaults are used, but it is possible to use one of the user sets instead. The boot-up configuration set is selected through the *Default Startup Set* feature. The same default set will be also applied after resetting the camera using the *Reset Device* command and even every time when opening the camera connection from the application.

```
LvHDevice hDev; // open the device to get the handle
// store current configuration in user set #1
LvSetEnum (hDev, LvDevice_UserSetSelector, LvUserSetSelector_UserSet1);
LvCmdExecute (hDev, LvDevice_UserSetSave);
// load and apply settings stored in user set #4
LvSetEnum (hDev, LvDevice_UserSetSelector, LvUserSetSelector_UserSet4);
LvCmdExecute (hDev, LvDevice_UserSetLoad);
// load and apply factory default settings
LvSetEnum (hDev, LvDevice_UserSetSelector, LvUserSetSelector_Default);
LvCmdExecute (hDev, LvDevice_UserSetLoad);
// use settings stored in user set #2 after during camera boot
LvSetEnum (hDev, LvDevice_UserSetDefaultSelector,
                 LvUserSetDefaultSelector_UserSet2);
```

GEN<i>CAM

```
// store current configuration in user set #1
write_enum (UserSetSelector, UserSet1)
exec_cmd (UserSetSave)
// load and apply settings stored in user set #4
write_enum (UserSetSelector, UserSet4)
exec_cmd (UserSetLoad)
// load and apply factory default settings
write_enum (UserSetSelector, Default)
exec_cmd (UserSetLoad)
// use settings stored in user set #2 after during camera boot
write_enum (UserSetDefaultSelector, UserSet2)
```

6.13.3. Software control of configuration sets

Besides the camera controlled configuration handling, the application might want to store/reload the configuration on its own. The configuration description stored by the application could be modified or reused for multiple compatible cameras.

Because of dependencies among individual features, however, the application has no way to know the correct order in which the features should be set to reach the desired goal reliably. The GenICam standard provides a solution for that, called “register streaming”, which is a standardized way to save/load full camera configuration. It was discussed before in Section 6.3.5, “Register streaming support” [p. 83]. Our cameras provide full support for register streaming.

Simplon API implements this functionality in `LvCameraSaveSettings()` and `LvCameraLoadSettings()` functions, see details in Simplon programmer's guide. Other GenICam compatible software packages should also provide similar functionality.

Besides that, the Simplon Source Code Generator (component of the Simplon Explorer) allows to record all feature settings, including executed commands. The recorded actions are provided by the Simplon Source Code Generator in form of source code that can be easily included in the user application.

6.13.4. Comparison of the two methods

Both methods of handling the user configuration sets, directly on the camera or by the application, have their advantages and disadvantages:

- The on-camera user set “travels” with the camera, independently on the application with which it is used. It is fully under camera control, so it is always correct — mistakes such as applying the user set to incompatible camera type or version are not possible.
- The camera's default user set is applied directly at camera boot, before any application even connects to it.
- The software controlled user sets can be stored once and applied to many cameras of the same type.
- The software controlled user sets might be edited by qualified personnel, although it must be done with maximum care.
- The action recorder/playback functionality can include command execution. On the other side, it strongly depends on the initial camera status.

6.14. External device control

The section documents features for camera's external device (peripheral) control. Peripheral devices can be controlled over a serial port interface.

6.14.1. Serial Port Control

The CheckSight/CheckSight-II cameras and some models of PicSight-GigE and PicSight-Smart are equipped with a serial port interface to control peripheral devices connected to the camera. The interface supports regular synchronous (command/response) string based communication.

All usual serial communication parameters can be configured:

Serial Port Baud Rate — Enumeration adjusting the baud rate. The values range from *2400* (`Baud2400`) to *115200* (`Baud115200`). The default value is *9600* (`Baud9600`).

Serial Port Parity — Enumeration adjusting the parity error detection method. Possible values are *None* (default value), *Odd* and *Even*.

Serial Port Data Bits — Enumeration adjusting data bits per character. Possible values are *8* (`DataBits8`, the default value) and *7* (`DataBits7`).

Serial Port Stop Bits — Enumeration adjusting stop bits per character. Possible values are *1* (`StopBits1`, the default value), *2* (`StopBits2`) and *1.5* (`StopBits1dot5`).

The interface always uses one start bit per character and no handshaking.

After sending the serial command, the camera has to wait for the response sent by the connected peripheral. The response timing is strongly device specific and thus wait operation can be controlled by following two parameters:

Serial Port Timeout — Maximal time the camera waits for incoming response characters before reporting the response to the application as floating point value in milliseconds. If no EOT marker is used, the full timeout is always spent before returning the response.

Serial Port EOT Marker — End of transmission marker — a string token (frequently just a single character) unequivocally identifying end of the response. Each device might use different characters to mark the end of transmission. Maximal length of the token is 8 characters, but the practically used markers are usually just 1—2 characters.

When specified, the camera observes the received characters and when the marker is encountered in the received buffer, the camera identifies the response as complete, stops waiting and reports the response immediately without completing the timeout.

If the camera receives a burst of characters containing the EOT marker, which is followed by additional characters, it waits little longer to see if another marker will arrive. It then reports the response until the last marker received.

If the EOT marker is not encountered in the response at all, the waiting stops as soon as the timeout expires.

Serial Port Max Response Length — Maximal expected response length. As soon as the camera receives the specified number of response characters, it immediately stops waiting and reports the response, regardless the timeout or EOT marker settings.

Value of `0` means that the camera does not wait for any response at all.

The default value is `128`, which is equal to the maximal supported length of the response string — meaning that the default value of this features does not impose any constraints on the expected response length.

The communication itself expects simple string based, synchronous messages (command followed by a single response):

1. Write the command message string to *Serial Port Command String*. Maximal supported command length is 128 characters.

2. Instruct the camera to send the command message to the serial port, executing the *Send Command Over Serial Port* command. The camera will send the characters and will start listening for the response.

3. Wait for completion of the command — wait until the “IsDone” GenICam property of the command yields `true` (in Simplon API this is inspected through `LvCmdIsDone()`). The camera signals the command as “done” when finished waiting for the response, according to the timeout and EOT marker configuration. Important: do not try to read the response before the command is completed, otherwise the communication may fail.

4. Check the command/response operation success — optional but recommended step. The status can be queried using the enumeration feature *Serial Port Command Status*. Possible values are *Success*, *Timeout* (depending on configuration this might signal a problem or not), *Port Busy* (e.g. when used by another application), *Communication Error* (generic), *Frame Error*, *Parity Error*, *Overflow*. If the status reports a failure, reading the response string itself would most probably yield irrelevant result.

5. Read the command response string from *Serial Port Command Response*. Maximal supported response length is 128 characters. Note that issuing a new command will discard any characters in the camera's communication input buffer and the camera will start waiting for a new response.

Copyright © Leutron Vision

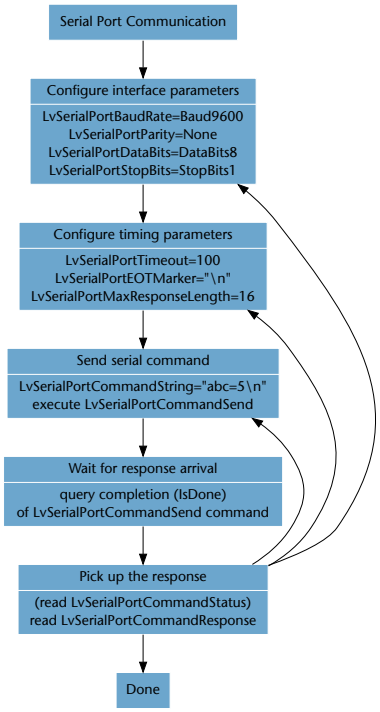


Figure 6.57. Flow chart: serial port communication

Simplon

```
LvHDevice hDev; // open the device to get the handle
// configure the communication interface parameters
LvSetEnum (hDev, LvDevice_LvSerialPortBaudRate,
                 LvSerialPortBaudRate_Baud9600);
LvSetEnum (hDev, LvDevice_LvSerialPortParity,
                 LvSerialPortParity_None);
LvSetEnum (hDev, LvDevice_LvSerialPortDataBits,
                 LvSerialPortDataBits_DataBits8);
LvSetEnum (hDev, LvDevice_LvSerialPortStopBits,
                 LvSerialPortStopBits_StopBits1);
// configure the response timing parameters
LvSetFloat (hDev, LvDevice_LvSerialPortTimeout, 100.0); // in ms
LvSetString (hDev, LvDevice_LvSerialPortEOTMarker, "\n");
LvSetInt32 (hDev, LvDevice_LvSerialPortMaxResponseLength, 16);
// send simple command 'abc=5\n'
LvSetString (hDev, LvDevice_LvSerialPortCommandString, "abc=5\n");
LvCmdExecute (hDev, LvDevice_LvSerialPortCommandSend);
// wait for the response and pick it up, unless failure is reported
// (wait until the operation end using LvCmdIsDone(),
// or by specifying the timeout in LvCmdExecute() call above)
int32_t EnumVal;
LvGetEnum (hDev, LvDevice_LvSerialPortCommandStatus, &EnumVal);
char StringVal[1024];
LvGetString (hDev, LvDevice_LvSerialPortCommandResponse, StringVal, 1024);
```

GEN<i>CAM

```
// configure the communication interface parameters
write_enum (LvSerialPortBaudRate, Baud9600)
write_enum (LvSerialPortParity, None)
write_enum (LvSerialPortDataBits, DataBits8)
write_enum (LvSerialPortStopBits, StopBits1)
// configure the response timing parameters
write_float (LvSerialPortTimeout, 100.0) // in ms
write_string (LvSerialPortEOTMarker, "\n")
write_int (LvSerialPortMaxResponseLength, 16)
// send simple command 'abc=5\n'
write_string (LvSerialPortCommandString, "abc=5\n")
exec_cmd (LvSerialPortCommandSend)
// wait for the response and pick it up, unless failure is reported
wait_until_done (LvSerialPortCommandSend)
read_enum (LvSerialPortCommandStatus)
read_string (LvSerialPortCommandResponse)
```

6.15. Special features

This section discusses special purpose features that do not belong to any other category or that span multiple categories.

Figure 6.58. Example GUI showing the "special" features

6.15.1. Special purpose triggers

Besides controlling the acquisition (Section 6.6, "Acquisition and trigger control" [p. 101]) and the I/O's (Section 6.7, "Digital I/O, counters and timers" [p. 113]), the external signals can be used to trigger also other specific actions. This can be configured through the "special purpose trigger" features.

The special purpose trigger configuration is very similar to the acquisition trigger features (Section 6.6, "Acquisition and trigger control" [p. 101]):

1. **Select trigger type.** Use the *Special Purpose Trigger Selector* feature to select the trigger type to be configured. In current firmware version, only the *Image Stamps Reset* trigger is available, but more types can be introduced in future.

2. **Trigger source (input line).** Configure the source of the trigger signals (*Special Purpose Trigger Source* feature). It can be a physical input line, such as *Line 1* (see Section 6.7, "Digital I/O, counters and timers" [p. 113], Section 3.8, "Input and output signals" [p. 32], Section 3.9, "Connector and cable description" [p. 37]). Unlike with the acquisition triggers, there is no "Software" option, issuing the trigger through executing the command feature *Generate Special Purpose Trigger* is always possible, regardless the actual trigger source configuration.

   Alternatively, the trigger source can be connected to one of the action commands, such as *Action 1*, if supported by the camera (see Section 6.8, "Action commands control" [p. 125]).

3. **Trigger activation (polarity).** The *Special Purpose Trigger Activation* feature controls which edge of the trigger input signal will be active, ie. which edge will actually ignite the trigger action. Possible values are *Rising Edge* and *Falling Edge*, rising edge being the transition

of signal level from logical `0` to logical `1`. The interpretation of signal levels for different input types (TTL/optocoupler) is discussed in Section 3.8, “Input and output signals” [p. 32]. Not applicable for software triggering.

4. **Configure additional options specific to given trigger.** If a given trigger action can be further controlled through additional features, do it now. For the *Image Stamps Reset* trigger it is possible to specify which stamps should be reset, see below.

5. **Triggering.** The triggers, being generated from external device using a trigger input line or from the application through the *Generate Special Purpose Trigger* command, can be issued at any suitable time.

6.15.1.1. Image stamps reset

The image stamps reset trigger can reset the “stamps” used to label the image buffers and other camera outputs (eg. the asynchronous events). After reset, given stamp starts counting again from zero. Following stamps are available:

Timestamp — Value of the 64-bit timestamp counter maintained by the device. The timestamp is used to label various data outputs provided by the camera, in particular the image buffers (through chunk data, Section 6.11, “Chunk data” [p. 146]) and the asynchronous events (Section 6.12, “Asynchronous events” [p. 151]).

Note that for PicSight-GigE cameras, the timestamp can be controlled also directly through the GigE Vision transport layer features (Section 6.4, “Transport layer control” [p. 84]).

Frame ID — The frame ID stamp serves as an image frame counter (also a 64-bit integer). It is optionally appended to every image buffer through the chunk data (Section 6.11, “Chunk data” [p. 146]).

The frame ID counter starts counting from zero when the camera boots up and keeps counting every acquired frame — until it is explicitly reset or until the value overflows the 64-bit range (in such case it would start counting from zero again).

Note that the counter does not count the image frames transferred to the host PC (the application), but rather all the frames internally acquired by the camera. If the camera boots in free running mode (default behavior), it starts internally acquiring and counting the frames (without transferring them to the application), even if the acquisition was not yet explicitly started by the application. This behaviour is particularly important for the PicSight-Smart cameras, which might be used in a “self-contained” mode, controlled by the smart application without intervention of a host PC.

Note that the frame ID is in general independent from the “block ID” low-level tags used by the GigE Vision protocol.

The image stamp reset trigger can be very useful to periodically synchronize the image stamps among multiple cooperating cameras. The trigger can be issued from an independent signal source or from one of the cameras, which works as a “master” in the group — such setup was demonstrated in Section 6.7.8, “Use for inter-camera synchronization” [p. 123].

To activate the image stamp reset trigger, it is first necessary to enable it for the individual stamps — selecting every desired stamp type in *Image Stamp Selector* and then activating the reset trigger functionality for it by setting the *Image Stamp Reset Enable* boolean feature. Each occurrence of the stamp resetting trigger will reset only those stamps that were explicitly enabled, leaving the others intact.

There's an alternative way that can be used to enable the stamps to be reset in one step — the *Image Stamps Reset Mask* feature, which is a 32-bit bitfield. The least significant bit controls the timestamp, the second bit controls the frame ID.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// configure timestamp reset trigger
LvSetEnum (hDev, LvDevice_LvSpecialPurposeTriggerSelector,
                 LvSpecialPurposeTriggerSelector_ImageStampsReset);
// configure hardware trigger input
LvSetEnum (hDev, LvDevice_LvSpecialPurposeTriggerSource,
                 LvSpecialPurposeTriggerSource_Line2); // Opto input
LvSetEnum (hDev, LvDevice_LvSpecialPurposeTriggerActivation,
                 LvSpecialPurposeTriggerActivation_RisingEdge);
// (or configure trigger over an action command)
LvSetEnum (hDev, LvDevice_LvSpecialPurposeTriggerSource,
                 LvSpecialPurposeTriggerSource_Action2);
// select the stamps to be reset
LvSetEnum (hDev, LvDevice_LvImageStampSelector,
                 LvImageStampSelector_Timestamp);
LvSetBool (hDev, LvDevice_LvImageStampResetEnable, 1);
LvSetEnum (hDev, LvDevice_LvImageStampSelector,
                 LvImageStampSelector_FrameID);
LvSetBool (hDev, LvDevice_LvImageStampResetEnable, 1);
// read back the stamps reset mask as a bitfield
int32_t IntVal;
LvGetInt32 (hDev, LvDevice_LvImageStampsResetMask, &IntVal);
// issue software trigger command to reset the timestamps
// (valid regardless the trigger source configuration)
LvCmdExecute (hDev, LvDevice_LvSpecialPurposeTriggerSoftware);
```

GEN<i>CAM

```
// configure triggered mode
// configure timestamp reset trigger
write_enum (LvSpecialPurposeTriggerSelector, ImageStampsReset)
// configure hardware trigger input
write_enum (LvSpecialPurposeTriggerSource, Line2) // Opto input
write_enum (LvSpecialPurposeTriggerActivation_RisingEdge, RisingEdge)
// (or configure trigger over an action command)
write_enum (LvSpecialPurposeTriggerSource, Action2)
// select the stamps to be reset
write_enum (LvImageStampSelector, Timestamp)
write_bool (LvImageStampResetEnable, true)
write_enum (LvImageStampSelector, FrameID)
write_bool (LvImageStampResetEnable, true)
// read back the stamps reset mask as a bitfield
read_int (LvImageStampsResetMask)
// issue software trigger command to reset the timestamps
// (valid regardless the trigger source configuration)
exec_cmd (LvSpecialPurposeTriggerSoftware)
```

6.15.2. Watchdog timer control

Some Leutron Vision camera models provide features to control the device's watchdog timer behavior. The watchdog timer is a subsystem allowing to monitor device health. When enabled, the

application needs to periodically reset the watchdog timer — the resets serve as the application "heartbeat". As long as each new heartbeat is issued withing a specified timeout duration, the watchdog stays silent. However, if the watchdog timer expires and the reset command is not issued, the watchdog assumes the application (or entire system) is locked and reboots the camera. The watchdog functionality is implemented in the FPGA, independently from the main camera control, so it remains functional even if the operating systems itself gets locked.

After the camera boots, the watchdog is always inactive. If desired, the application has to enable it explicitly (*Watchdog Enable*), specifying the desired *Watchdog Timeout Duration*. Once enabled, the application is responsible issuing the *Watchdog Reset* signals in period lower than the configured timeout to prevent the watchdog timeout expiration and device reboot. The watchdog can be disabled again. Furthermore, the current *Watchdog Timer Value* (time remaining until expiration) can be queried.

It is important that the application always disables the watchdog timer whenever it intends to stop updating it regularly, such as when the application shuts down — otherwise the system would be rebooted as soon as the timer expires. The watchdog timer is never disabled automatically.

Finally, the integer *Watchdog Failure Count* provides information about how many watchdog failure reboots happened in the device lifetime.

Note that all the watchdog related features are in "guru" feature visibility level.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// possibly read number of watchdog failures in device's lifetime
LvGetInt32 (hDev, LvDevice_LvWatchdogFailureCount, &IntVal);
// configure and enable the watchdog timer...
LvSetFloat (hDev, LvDevice_LvWatchdogTimerDuration, 20.0); // in seconds
LvSetBool (hDev, LvDevice_LvWatchdogEnable, 1);
// ...and immediatelly start feeding it with regular reset signals
LvCmdExecute (hDev, LvDevice_LvWatchdogTimerReset);
// the actual value of the watchdog timer can be queried
double FloatVal;
LvGetFloat (hDev, LvDevice_LvWatchdogTimerValue, &FloatVal);
// when not needed any more, disable the watchdog timer
LvSetBool (hDev, LvDevice_LvWatchdogEnable, 0);
```

GEN<i>CAM

```
// possibly read number of watchdog failures in device's lifetime
read_int (LvWatchdogFailureCount)
// configure and enable the watchdog timer...
write_float (LvWatchdogTimerDuration, 20.0) // in seconds
write_bool (LvWatchdogEnable, true)
// ...and immediatelly start feeding it with regular reset signals
exec_cmd (LvWatchdogTimerReset)
// the actual value of the watchdog timer can be queried
read_float (LvWatchdogTimerValue)
// when not needed any more, disable the watchdog timer
write_bool (LvWatchdogEnable, false)
```

6.16. Feature summary

The following table provides condensed summary of all the features existing among all our camera families. Some of the features are only supported by certain camera families, some of them even just by certain few camera models. The range of allowed values might differ.

For detailed explanation of the purpose of each feature group and their dependencies, refer to previous sections in this chapter. To learn availability of a feature on a particular camera model, refer to the model datasheet or contact our sales department.

Name	Description
Feature category: Device Control , refer to Section 6.3, “Information about the camera” [p. 80]	
`DeviceVendorName` *GUI name: Vendor Name*	Name of the manufacturer of the device Type: string GUI visibility: beginner Read-only feature
`DeviceModelName` *GUI name: Model Name*	Model name of the device Type: string GUI visibility: beginner Read-only feature
`DeviceManufacturerInfo` *GUI name: Manufacturer Info*	Manufacturer information about the device Type: string GUI visibility: beginner Read-only feature
`DeviceVersion` *GUI name: Device Hardware Version*	Version of the device Type: string GUI visibility: beginner Read-only feature
`DeviceFirmwareVersion` *GUI name: Device Firmware Version*	Version of the device firmware Type: string GUI visibility: beginner Read-only feature
`LvRecoveryFirmwareVersion` *GUI name: Recovery Firmware Version*	Indicates the version of the firmware and software to which the device would recover Type: string GUI visibility: guru Read-only feature
`DeviceID` *GUI name: Device ID*	Device identifier (serial number) Type: string GUI visibility: expert Read-only feature
`DeviceUserID` *GUI name: Device User ID*	User-programmable device identifier Type: string GUI visibility: beginner

Name	Description
`LvSensorID` *GUI name: Sensor ID*	Serial number of the sensor board Type: string GUI visibility: guru Read-only feature
`LvGrabberID` *GUI name: Grabber ID*	Serial number of the grabber board Type: string GUI visibility: guru Read-only feature
`DeviceScanType` *GUI name: Device Scan Type*	Scan type of the sensor. Type: enumeration Possible values: `Areascan` GUI visibility: expert Read-only feature
`DeviceRegistersStreamingStart` *GUI name: Device Registers Streaming Start*	Announces start of register streaming Type: command GUI visibility: guru
`DeviceRegistersStreamingEnd` *GUI name: Device Registers Streaming End*	Announces end of register streaming Type: command GUI visibility: guru
`DeviceRegistersCheck` *GUI name: Device Registers Check*	Requests validation of the register set Type: command GUI visibility: expert
`DeviceRegistersValid` *GUI name: Device Registers Valid*	Result of registers validity check Type: bool GUI visibility: expert Read-only feature
`DeviceReset` *GUI name: Reset Device*	Resets the device Type: command GUI visibility: guru
`DeviceClockSelector` *GUI name: Clock Selector*	Selects a device clock frequency Type: enumeration Possible values: `SensorDigitization` GUI visibility: expert
`DeviceClockFrequency` [`DeviceClockSelector`] *GUI name: Clock Frequency*	Frequency of the selected clock Type: float GUI visibility: expert Streamable feature, stored with user configuration sets Unit: Hz

Name	Description
`DeviceTemperatureSelector` *GUI name: Temperature Selector*	Selects a temperature measurement location Type: enumeration Possible values: `Sensor`, `Mainboard` GUI visibility: expert
`DeviceTemperature` [`DeviceTemperatureSelector`] *GUI name: Temperature*	Current temperature at the selected location Type: float GUI visibility: expert Read-only feature Unit: C
`LvDeviceUpTime` *GUI name: Up Time*	Current up-time of the device Type: integer GUI visibility: expert Unit: ms
`LvDeviceType` *GUI name: Device Type*	Indicates the basic type of the device Type: string GUI visibility: expert Read-only feature
`DeviceSFNCVersionMajor` *GUI name: Device SFNC Version Major*	Major version of SFNC standard used by the device Type: integer GUI visibility: beginner Read-only feature
`DeviceSFNCVersionMinor` *GUI name: Device SFNC Version Minor*	Minor version of SFNC standard used by the device Type: integer GUI visibility: beginner Read-only feature
`DeviceSFNCVersionSubMinor` *GUI name: Device SFNC Version Subminor*	Sub-minor version of SFNC standard used by the device Type: integer GUI visibility: beginner Read-only feature
Feature category: Image Format Control , refer to Section 6.5, “Image format” [p. 85]	
`SensorWidth` *GUI name: Sensor Width*	Effective width of the sensor in pixels. Type: integer GUI visibility: expert Read-only feature Unit: px

Name	Description
`SensorHeight` *GUI name: Sensor Height*	Effective height of the sensor in pixels. Type: integer GUI visibility: expert Read-only feature Unit: px
`WidthMax` *GUI name: Max Width*	Maximum width of the image. Type: integer GUI visibility: expert Read-only feature Unit: px
`HeightMax` *GUI name: Max Height*	Maximum height of the image. Type: integer GUI visibility: expert Read-only feature Unit: px
`Width` *GUI name: Width*	Image width provided by the device Type: integer GUI visibility: beginner Streamable feature, stored with user configuration sets Unit: px
`Height` *GUI name: Height*	Image height provided by the device Type: integer GUI visibility: beginner Streamable feature, stored with user configuration sets Unit: px
`OffsetX` *GUI name: X Offset*	Horizontal offset from the origin of the AOI. Type: integer GUI visibility: beginner Streamable feature, stored with user configuration sets Unit: px
`OffsetY` *GUI name: Y Offset*	Vertical offset from the origin of the AOI. Type: integer GUI visibility: beginner Streamable feature, stored with user configuration sets Unit: px

Name	Description
`PixelFormat` *GUI name: Pixel Format*	Pixel format provided by the device Type: enumeration Possible values: `Mono8`, `Mono8Signed`, `Mono10`, `Mono10Packed`, `Mono12`, `Mono12Packed`, `Mono16`, `BayerGR8`, `BayerRG8`, `BayerGB8`, `BayerBG8`, `BayerGR10`, `BayerRG10`, `BayerGB10`, `BayerBG10`, `BayerGR12`, `BayerRG12`, `BayerGB12`, `BayerBG12`, `RGB8Packed`, `BGR8Packed`, `RGBA8Packed`, `BGRA8Packed` GUI visibility: beginner Streamable feature, stored with user configuration sets
`BinningHorizontal` *GUI name: Horizontal Binning*	Horizontal binning factor Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets
`BinningVertical` *GUI name: Vertical Binning*	Vertical binning factor Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets
`DecimationHorizontal` *GUI name: Horizontal Decimation*	Horizontal decimation factor Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets
`DecimationVertical` *GUI name: Vertical Decimation*	Vertical decimation factor Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets
`LvAOIMode` *GUI name: AOI Mode*	Mode of controlling the area of interest Type: enumeration Possible values: `Automatic`, `ClipOnTransfer`, `Manual` GUI visibility: expert Streamable feature, stored with user configuration sets

Name	Description
`LvReadoutWidth` *GUI name: Readout Width*	Width of the sensor-side area of interest in pixels Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets Unit: px
`LvReadoutHeight` *GUI name: Readout Height*	Height of the sensor-side area of interest in pixels Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets Unit: px
`LvReadoutOffsetX` *GUI name: Readout X Offset*	X offset (left offset) of the sensor-side area of interest in pixels Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets Unit: px
`LvReadoutOffsetY` *GUI name: Readout Y Offset*	Y offset (top offset) of the sensor-side area of interest in pixels Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets Unit: px
`LvVariablePayloadSize` *GUI name: Variable Payload Size*	Flag switching on the variable payload size mode Type: bool GUI visibility: expert Streamable feature, stored with user configuration sets
Feature category: Acquisition Control , refer to Section 6.6, “Acquisition and trigger control” [p. 101]	
`AcquisitionMode` *GUI name: Acquisition Mode*	Acquisition mode of the device Type: enumeration Possible values: `SingleFrame`, `MultiFrame`, `Continuous` GUI visibility: beginner Streamable feature, stored with user configuration sets
`AcquisitionStart` *GUI name: Acquisition Start*	Starts image acquisition Type: command GUI visibility: beginner

Name	Description
`AcquisitionStop` *GUI name: Acquisition Stop*	Stops image acquisition Type: command GUI visibility: beginner
`AcquisitionArm` *GUI name: Acquisition Arm*	Arms the camera for acquisition start Type: command GUI visibility: expert
`TriggerSelector` *GUI name: Trigger Selector*	Selects the type of trigger to configure Type: enumeration Possible values: `FrameStart` GUI visibility: beginner
`TriggerMode` [`TriggerSelector`] *GUI name: Trigger Mode*	Controls if the selected trigger is active Type: enumeration Possible values: `Off`, `On` GUI visibility: beginner Streamable feature, stored with user configuration sets
`TriggerSoftware` [`TriggerSelector`] *GUI name: Generate Software Trigger*	Generates a software trigger Type: command GUI visibility: beginner
`TriggerSource` [`TriggerSelector`] *GUI name: Trigger Source*	Signal source for the selected trigger Type: enumeration Possible values: `Line1`, `Line2`, `Line3`, `Line4`, `Line5`, `Line6`, `Line7`, `Line8`, `Line17`, `Line18`, `Line19`, `Line20`, `Line21`, `Line22`, `Line23`, `Line24`, `Action1`, `Action2`, `Action3`, `Action4`, `Action5`, `Action6`, `Action7`, `Action8`, `Software` GUI visibility: beginner Streamable feature, stored with user configuration sets
`TriggerActivation` [`TriggerSelector`] *GUI name: Trigger Activation*	Activation mode of the trigger. Type: enumeration Possible values: `RisingEdge`, `FallingEdge` GUI visibility: beginner Streamable feature, stored with user configuration sets
`TriggerDelay` [`TriggerSelector`] *GUI name: TriggerDelay*	Trigger delay in microseconds Type: float GUI visibility: expert Streamable feature, stored with user configuration sets Unit: us

Name	Description
`TriggerDivider` [`TriggerSelector`] *GUI name: Trigger Divider*	Trigger divider Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets
`LvTriggerCaching` [`TriggerSelector`] *GUI name: Trigger Caching*	Caching mode for the selected trigger Type: enumeration Possible values: `Cache`, `Drop` GUI visibility: expert Streamable feature, stored with user configuration sets
`ExposureMode` *GUI name: Exposure Mode*	Exposure (shutter) mode Type: enumeration Possible values: `Timed` GUI visibility: beginner Streamable feature, stored with user configuration sets
`LvLongRangeExposureMode` *GUI name: Long Range Exposure Mode*	Switches to mode with wider range of exposure times Type: bool GUI visibility: beginner Streamable feature, stored with user configuration sets
`LvGlobalResetMode` *GUI name: Global Reset Mode*	Switches to mode with wider range of exposure times Type: bool GUI visibility: beginner Streamable feature, stored with user configuration sets
`ExposureTime` *GUI name: Exposure Time*	Exposure time in microseconds Type: float GUI visibility: beginner Streamable feature, stored with user configuration sets Unit: us
`ExposureAuto` *GUI name: Automatic Exposure*	Selects the automatic exposure mode Type: enumeration Possible values: `Off`, `Once`, `Continuous` GUI visibility: beginner Streamable feature, stored with user configuration sets

Name	Description
`LvAcquisitionFrameRateControlMode` *GUI name: Acquisition Frame Rate Control*	Switches the acquisition frame rate control on or off Type: enumeration Possible values: `Off`, `On` GUI visibility: beginner Streamable feature, stored with user configuration sets
`AcquisitionFrameRate` *GUI name: Acquisition Frame Rate*	Acquisition frame rate in frames per second (Hz) Type: float GUI visibility: beginner Streamable feature, stored with user configuration sets Unit: Hz
Feature category: Digital I/O Control , refer to Section 6.7, "Digital I/O, counters and timers" [p. 113]	
`LineSelector` *GUI name: Line Selector*	Selects the I/O line for querying and configuration Type: enumeration Possible values: `Line1`, `Line2`, `Line3`, `Line4`, `Line5`, `Line6`, `Line7`, `Line8`, `Line9`, `Line10`, `Line11`, `Line12`, `Line13`, `Line14`, `Line15`, `Line16`, `Line17`, `Line18`, `Line19`, `Line20`, `Line21`, `Line22`, `Line23`, `Line24`, `Line25`, `Line26`, `Line27`, `Line28`, `Line29`, `Line30`, `Line31`, `Line32` GUI visibility: expert
`LineMode` [`LineSelector`] *GUI name: Line Mode*	Line mode (input or output) Type: enumeration Possible values: `Input`, `Output` GUI visibility: expert Read-only feature
`LineFormat` [`LineSelector`] *GUI name: Line Format*	Sets the electrical configuration of the selected line Type: enumeration Possible values: `NoConnect`, `TriState`, `TTL`, `LVDS`, `RS422`, `OptoCoupled` GUI visibility: expert Read-only feature

Name	Description
`LineSource` [`LineSelector`] *GUI name: Line Source*	Internal signal driving the line's ouptut Type: enumeration Possible values: `Off`, `ExposureActive`, `Timer1Active`, `Timer2Active`, `Timer3Active`, `Timer4Active`, `UserOutput1`, `UserOutput2`, `UserOutput3`, `UserOutput4`, `UserOutput5`, `UserOutput6`, `UserOutput7`, `UserOutput8` GUI visibility: expert Streamable feature, stored with user configuration sets
`LineInverter` [`LineSelector`] *GUI name: Line Inverter*	Inverts the signal Type: bool GUI visibility: expert Streamable feature, stored with user configuration sets
`LvLineDebounceDuration` [`LineSelector`] *GUI name: Line Debounce Duration*	Duration of line debounce period Type: float GUI visibility: expert Streamable feature, stored with user configuration sets Unit: us
`LineStatus` [`LineSelector`] *GUI name: Line Status*	Status of the selected line Type: bool GUI visibility: expert Read-only feature
`LineStatusAll` *GUI name: Line Status All*	Status of i/o lines Type: integer GUI visibility: expert Read-only feature
`UserOutputSelector` *GUI name: User Output Selector*	Selects the user output for querying and configuration Type: enumeration Possible values: `UserOutput1`, `UserOutput2`, `UserOutput3`, `UserOutput4`, `UserOutput5`, `UserOutput6`, `UserOutput7`, `UserOutput8` GUI visibility: expert

Name	Description
`UserOutputValue` [`UserOutputSelector`] *GUI name: User Output Value*	Status of the selected user output Type: bool GUI visibility: expert Streamable feature, stored with user configuration sets
`UserOutputValueAll` *GUI name: User Output Value All*	Status of user outputs Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets
`UserOutputValueAllMask` *GUI name: User Output Value All Mask*	Mask for the User Output Value All bitfield Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets
Feature category: Counter and Timer Control , refer to Section 6.7, “Digital I/O, counters and timers” [p. 113]	
`CounterSelector` *GUI name: Counter Selector*	Selects which counter to configure Type: enumeration Possible values: `Counter1`, `Counter2`, `Counter3`, `Counter4` GUI visibility: expert
`LvCounterMode` [`CounterSelector`] *GUI name: Counter Mode*	Selects working mode of the selected counter. Type: enumeration Possible values: `Autoreset` GUI visibility: expert Streamable feature, stored with user configuration sets
`CounterEventSource` [`CounterSelector`] *GUI name: Counter Event Source*	Signal incrementing the counter Type: enumeration Possible values: `Off`, `FrameTrigger` GUI visibility: expert Streamable feature, stored with user configuration sets
`CounterReset` [`CounterSelector`] *GUI name: Counter Reset*	Resets the selected counter Type: command GUI visibility: expert
`CounterValue` [`CounterSelector`] *GUI name: Counter Value*	Current value of the selected counter Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets

Name	Description
`CounterDuration` [`CounterSelector`] *GUI name: Counter Duration*	Duration (number of events) for the selected counter Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets
`TimerSelector` *GUI name: Timer Selector*	Selects which timer to configure Type: enumeration Possible values: `Timer1`, `Timer2`, `Timer3`, `Timer4` GUI visibility: expert
`TimerDuration` [`TimerSelector`] *GUI name: Timer Duration*	Duration of the selected timer Type: float GUI visibility: expert Streamable feature, stored with user configuration sets Unit: us
`TimerDelay` [`TimerSelector`] *GUI name: Timer Delay*	Delay for the selected timer Type: float GUI visibility: expert Streamable feature, stored with user configuration sets Unit: us
`TimerTriggerSource` [`TimerSelector`] *GUI name: Timer Trigger Source*	Signal activating the timer Type: enumeration Possible values: `Off`, `FrameTrigger`, `Counter1End`, `Counter2End`, `Counter3End`, `Counter4End`, `UserOutput1`, `UserOutput2`, `UserOutput3`, `UserOutput4`, `UserOutput5`, `UserOutput6`, `UserOutput7`, `UserOutput8` GUI visibility: expert Streamable feature, stored with user configuration sets
Feature category: Action Control , refer to Section 6.8, “Action commands control” [p. 125]	
`ActionDeviceKey` *GUI name: Action Device Key*	Provides the device key that allows the device to check the validity of action commands. Type: integer GUI visibility: guru
`ActionSelector` *GUI name: Action Selector*	Selects to which action signal further action settings apply. Type: integer GUI visibility: guru

Name	Description
`ActionGroupKey` [`ActionSelector`] *GUI name: Action Group Key*	Provides the key that the device will use to validate the action on reception of the action protocol message. Type: integer GUI visibility: guru
`ActionGroupMask` [`ActionSelector`] *GUI name: Action Group Mask*	Provides the mask that the device will use to validate the action on reception of the action protocol message. Type: integer GUI visibility: guru
Feature category: Special Features , refer to Section 6.15, “Special features” [p. 163]	
`LvSpecialPurposeTriggerSelector` *GUI name: Special Purpose Trigger Selector*	Selects the special purpose trigger type to configure Type: enumeration Possible values: `ImageStampsReset` GUI visibility: expert
`LvSpecialPurposeTriggerSource` [`LvSpecialPurposeTriggerSelector`] *GUI name: Special Purpose Trigger Source*	Signal source for the selected trigger Type: enumeration Possible values: `Off`, `Line1`, `Line2`, `Line3`, `Line4`, `Line5`, `Line6`, `Line7`, `Line8`, `Line17`, `Line18`, `Line19`, `Line20`, `Line21`, `Line22`, `Line23`, `Line24`, `Action1`, `Action2`, `Action3`, `Action4`, `Action5`, `Action6`, `Action7`, `Action8` GUI visibility: expert Streamable feature, stored with user configuration sets
`LvSpecialPurposeTriggerActivation` [`LvSpecialPurposeTriggerSelector`] *GUI name: Special Purpose Trigger Activation*	Activation mode of the trigger. Type: enumeration Possible values: `RisingEdge`, `FallingEdge` GUI visibility: expert Streamable feature, stored with user configuration sets
`LvSpecialPurposeTriggerSoftware` [`LvSpecialPurposeTriggerSelector`] *GUI name: Generate Special Purpose Trigger*	Generates a software trigger Type: command GUI visibility: expert
`LvImageStampsResetMask` *GUI name: Image Stamps Reset Mask*	A single bitfield that selects which features will be reset by the image stamp reset trigger Type: integer GUI visibility: guru
`LvImageStampSelector` *GUI name: Image Stamp Selector*	Selects an image stamp type for configuration Type: enumeration Possible values: `Timestamp`, `FrameID` GUI visibility: expert

Name	Description
`LvImageStampResetEnable` [`LvImageStampSelector`] *GUI name: Image Stamp Reset Enable*	Enables/disables the reset trigger functionality for the selected image stamp type Type: bool GUI visibility: expert Streamable feature, stored with user configuration sets
`LvBootSwitch` *GUI name: Boot Switch*	Selects the firmware type to load on next boot Type: enumeration Possible values: `PureGEV`, `Legacy` GUI visibility: guru
`LvBayerDecoderAlgorithm` *GUI name: Bayer Decoder Algorithm*	Selects the Bayer decoder algorithm Type: enumeration Possible values: `VariableGradient` GUI visibility: expert Streamable feature, stored with user configuration sets
`LvBayerDecoderThreshold` *GUI name: Bayer Decoder Threshold*	Sets the Bayer decoder threshold Type: float GUI visibility: guru Streamable feature, stored with user configuration sets
`LvWatchdogEnable` *GUI name: Watchdog Enable*	Enables the watchdog reset function Type: bool GUI visibility: guru Streamable feature, stored with user configuration sets
`LvWatchdogTimerDuration` *GUI name: Watchdog Timer Duration*	When watchdog is enabled, the device reboots when the timeout specified expires Type: float GUI visibility: guru Streamable feature, stored with user configuration sets Unit: s
`LvWatchdogTimerValue` *GUI name: Watchdog Timer Value*	Current watchdog timer value - reports the current value, after which the timer expires Type: float GUI visibility: guru Read-only feature Unit: s
`LvWatchdogTimerReset` *GUI name: Watchdog Timer Reset*	Resets the watchdog timer, the watchdog starts counting the specified timeout again Type: command GUI visibility: guru

Name	Description
`LvWatchdogFailed` *GUI name: Watchdog Failed*	Signals that the last device reboot was initiated by the watchdog function Type: bool GUI visibility: guru
`LvWatchdogFailureCount` *GUI name: Watchdog Failure Count*	Number of watchdog failures (reboots) in camera's lifetime Type: integer GUI visibility: guru
Feature category: Analog Control , refer to Section 6.9, “Analog controls” [p. 127]	
`GainSelector` *GUI name: Gain Selector*	Selects which gain type to configure Type: enumeration Possible values: `All`, `AnalogAll`, `DigitalAll` GUI visibility: beginner
`Gain` [`GainSelector`] *GUI name: Gain*	Gain value for the selected gain type Type: float GUI visibility: beginner Streamable feature, stored with user configuration sets Unit: dB
`GainAuto` [`GainSelector`] *GUI name: Automatic Gain*	Automatic gain mode Type: enumeration Possible values: `Off`, `Once`, `Continuous` GUI visibility: beginner Streamable feature, stored with user configuration sets
`BlackLevelSelector` *GUI name: Black Level Selector*	Selects which black level type to configure Type: enumeration Possible values: `All` GUI visibility: expert
`BlackLevel` [`BlackLevelSelector`] *GUI name: Black Level*	Black level value Type: float GUI visibility: expert Streamable feature, stored with user configuration sets Unit: DN
`BlackLevelAuto` [`BlackLevelSelector`] *GUI name: Automatic Black Level*	Automatic black level mode Type: enumeration Possible values: `Off`, `Once`, `Continuous` GUI visibility: expert Streamable feature, stored with user configuration sets

Name	Description
Feature category: Color Transformation Control , refer to Section 6.10, “Luminance and color handling” [p. 134]	
`ColorTransformationSelector` *GUI name: Color Transformation Selector*	Selects which color transformation module is controlled by the color transformation features. Type: enumeration Possible values: `RGBtoRGB` GUI visibility: expert
`ColorTransformationEnable` [`ColorTransformationSelector`] *GUI name: Color Transformation Enable*	Activates the selected Color Transformation module Type: bool GUI visibility: expert Streamable feature, stored with user configuration sets
`ColorTransformationValueSelector` [`ColorTransformationSelector`] *GUI name: Color Transformation Value Selector*	Selects the color transformation matrix entry Type: enumeration Possible values: `Gain00`, `Gain01`, `Gain02`, `Gain10`, `Gain11`, `Gain12`, `Gain20`, `Gain21`, `Gain22` GUI visibility: expert
`ColorTransformationValue` [`ColorTransformationValueSelector`] *GUI name: Color Transformation Entry Value*	Value of the selected color transformation matrix entry Type: float GUI visibility: expert Streamable feature, stored with user configuration sets
Feature category: External Device Control , refer to Section 6.14, “External device control” [p. 160]	
`LvExternalDeviceControlMode` *GUI name: External Device Control Mode*	Selects the operation mode of external device control. Type: enumeration Possible values: `Custom` GUI visibility: expert Streamable feature, stored with user configuration sets
`LvExternalADCSelector` *GUI name: External ADC Selector*	External ADC to configure. Type: enumeration Possible values: `ExternalADC1`, `ExternalADC2`, `ExternalADC3`, `ExternalADC4` GUI visibility: expert

Name	Description
`LvExternalADCValue` [`LvExternalADCSelector`] *GUI name: External ADC Value*	Value of the selected external ADC Type: integer GUI visibility: expert Read-only feature Unit: DN
`LvPowerSwitchCurrentAction` *GUI name: Power Switch Current Action*	Action currently performed by a power switch Type: enumeration Possible values: `Idle`, `Pulse`, `Calibrate`, `AdjustPosition`, `Drive` GUI visibility: expert Read-only feature
`LvPowerSwitchSelector` *GUI name: Power Switch Selector*	Power switch to configure. Type: enumeration Possible values: `PowerSwitch1`, `PowerSwitch2`, `PowerSwitch3`, `PowerSwitch4` GUI visibility: expert
`LvPowerSwitchBoundADC` [`LvPowerSwitchSelector`] *GUI name: Power Switch Bound ADC*	Binds an external ADC to the power switch Type: enumeration Possible values: `None`, `ExternalADC1`, `ExternalADC2`, `ExternalADC3`, `ExternalADC4` GUI visibility: expert Streamable feature, stored with user configuration sets
`LvPowerSwitchDrive` [`LvPowerSwitchSelector`] *GUI name: Power Switch Drive*	Drives the selected power switch with desired polarity Type: enumeration Possible values: `Off`, `Plus`, `Minus` GUI visibility: expert
`LvPowerSwitchPulseDuration` [`LvPowerSwitchSelector`] *GUI name: Power Switch Pulse Duration*	Power switch pulse duration in microseconds Type: float GUI visibility: expert Streamable feature, stored with user configuration sets Unit: us
`LvPowerSwitchPulsePlus` [`LvPowerSwitchSelector`] *GUI name: Power Switch Pulse Plus*	Pulses the selected power switch with plus polarity Type: command GUI visibility: expert

Name	Description
`LvPowerSwitchPulseMinus` [`LvPowerSwitchSelector`] *GUI name: Power Switch Pulse Minus*	Pulses the selected power switch with minus polarity Type: command GUI visibility: expert
`LvLensControlMinCalibrationRange` [`LvPowerSwitchSelector`] *GUI name: Lens Control Min Calibration Range*	Minimum ADC range required for calibration Type: integer GUI visibility: guru Streamable feature, stored with user configuration sets
`LvLensControlCalibrate` [`LvPowerSwitchSelector`] *GUI name: Lens Control Calibrate*	Starts automatic calibration Type: command GUI visibility: expert
`LvLensControlCalibrationStatus` [`LvPowerSwitchSelector`] *GUI name: Lens Control Calibration Status*	Reports current calibration status of the selected power switch and its bound ADC Type: enumeration Possible values: `Invalid`, `Valid` GUI visibility: expert Read-only feature
`LvLensControlMinusEnd` [`LvPowerSwitchSelector`] *GUI name: Lens Control Minus End*	Calibrated minimal ADC achievable by driving the power switch's with minus polarity (plus if the polarity is inverted) Type: integer GUI visibility: guru Streamable feature, stored with user configuration sets
`LvLensControlPlusEnd` [`LvPowerSwitchSelector`] *GUI name: Lens Control Plus End*	Calibrated maximal ADC achievable by driving the power switch's with plus polarity (minus if the polarity is inverted) Type: integer GUI visibility: guru Streamable feature, stored with user configuration sets
`LvLensControlInvertedPolarity` [`LvPowerSwitchSelector`] *GUI name: Lens Control Inverted Polarity*	Indicates if the lens is wired with inverted polarity Type: bool GUI visibility: guru Streamable feature, stored with user configuration sets
`LvLensControlPulsePeriod` [`LvPowerSwitchSelector`] *GUI name: Lens Control Pulse Period*	Calibrated slow motion pulse period for the selected power switch, in microsesonds Type: float GUI visibility: guru Streamable feature, stored with user configuration sets Unit: us

Name	Description
`LvLensControlDutyCycle` [`LvPowerSwitchSelector`] *GUI name: Lens Control Duty Cycle*	Calibrated slow motion duty cycle on the power switch Type: integer GUI visibility: guru Streamable feature, stored with user configuration sets Unit: %
`LvLensControlTargetApproach` [`LvPowerSwitchSelector`] *GUI name: Lens Control Target Approach*	Selects how the target lens position should be approached Type: enumeration Possible values: `Direct`, `FromPlus`, `FromMinus` GUI visibility: expert Streamable feature, stored with user configuration sets
`LvLensControlNrSlowSteps` [`LvPowerSwitchSelector`] *GUI name: Lens Control Nr Slow Steps*	Number of slow steps before the target position Type: integer GUI visibility: guru Streamable feature, stored with user configuration sets
`LvLensControlTargetPosition` [`LvPowerSwitchSelector`] *GUI name: Lens Control Target Position*	Target lens position Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets
`LvLensControlAdjustPosition` [`LvPowerSwitchSelector`] *GUI name: Lens Control Adjust Position*	Adjusts the target lens position Type: command GUI visibility: expert
`LvLensControlCalibrateAll` *GUI name: Lens Control Calibrate All*	Starts automatic calibration Type: command GUI visibility: expert
`LvSerialPortBaudRate` *GUI name: Serial Port Baud Rate*	Baud rate used for the serial port communication Type: enumeration Possible values: `Baud2400`, `Baud4800`, `Baud9600`, `Baud14400`, `Baud19200`, `Baud38400`, `Baud57600`, `Baud115200` GUI visibility: expert Streamable feature, stored with user configuration sets

Name	Description
`LvSerialPortParity` *GUI name: Serial Port Parity*	Parity used for the serial port communication Type: enumeration Possible values: `None`, `Odd`, `Even` GUI visibility: expert Streamable feature, stored with user configuration sets
`LvSerialPortDataBits` *GUI name: Serial Port Data Bits*	Data bits per character for the serial port communication Type: enumeration Possible values: `DataBits7`, `DataBits8` GUI visibility: expert Streamable feature, stored with user configuration sets
`LvSerialPortStopBits` *GUI name: Serial Port Stop Bits*	Stop bits per character for the serial port communication Type: enumeration Possible values: `StopBits1`, `StopBits1dot5`, `StopBits2` GUI visibility: expert Streamable feature, stored with user configuration sets
`LvSerialPortTimeout` *GUI name: Serial Port Timeout*	Timeout value used to finish waiting for command response Type: float GUI visibility: expert Streamable feature, stored with user configuration sets Unit: ms
`LvSerialPortEOTMarker` *GUI name: Serial Port EOT Marker*	Short string (or single character) marking end of transmission Type: string GUI visibility: expert Streamable feature, stored with user configuration sets
`LvSerialPortMaxResponseLength` *GUI name: Serial Port Max Response Length*	Maximal expected length of the command response Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets Unit: B

Name	Description
`LvSerialPortCommandString` *GUI name: Serial Port Command String*	String of the ASCII command to be sent over the serial port Type: string GUI visibility: expert
`LvSerialPortCommandSend` *GUI name: Send Command Over Serial Port*	Sends the ASCII command over the serial port Type: command GUI visibility: expert
`LvSerialPortCommandResponse` *GUI name: Serial Port Command Response*	Response to the ASCII command sent over the serial port Type: string GUI visibility: expert Read-only feature
`LvSerialPortCommandStatus` *GUI name: Serial Port Command Status*	Status code indicating success of the last command Type: enumeration Possible values: `Success`, `Timeout`, `PortBusy`, `CommunicationError`, `FrameError`, `ParityError`, `Overflow` GUI visibility: expert Read-only feature
Feature category: LUT Control and Image Adjustment , refer to Section 6.10, “Luminance and color handling” [p. 134]	
`LvLUTMode` *GUI name: LUT Mode*	Selects the LUT control mode Type: enumeration Possible values: `Direct`, `BalanceWhite`, `BrightnessContrast` GUI visibility: expert Streamable feature, stored with user configuration sets
`LUTSelector` *GUI name: LUT Selector*	Selects which LUT to configure Type: enumeration Possible values: `Luminance`, `Red`, `Green`, `Blue` GUI visibility: expert
`LUTEnable` [`LUTSelector`] *GUI name: LUT Enable*	Activates the selected LUT Type: bool GUI visibility: expert Streamable feature, stored with user configuration sets
`LUTIndex` [`LUTSelector`] *GUI name: LUT Index*	LUT element index Type: integer GUI visibility: guru

Name	Description
`LUTValue` [`LUTIndex`] *GUI name: LUT Value*	LUT value for the selected index Type: integer GUI visibility: guru Streamable feature, stored with user configuration sets
`LUTValueAll` [`LUTSelector`] *GUI name: LUT Value All*	Accesses the entire content of the selected LUT in one chunk access Type: raw register GUI visibility: guru
`LvLUTReset` [`LUTSelector`] *GUI name: LUT Reset*	Resets the LUT settings Type: command GUI visibility: expert
`BalanceRatioSelector` *GUI name: Balance Ratio Selector*	Selects which color channel to configure Type: enumeration Possible values: `Red`, `Green`, `Blue` GUI visibility: expert
`BalanceRatio` [`BalanceRatioSelector`] *GUI name: Balance Ratio*	White balance ratio coefficient Type: float GUI visibility: expert Streamable feature, stored with user configuration sets
`BalanceWhiteAuto` *GUI name: Automatic White Balance*	Controls the mode for automatic white balancing between the color channels. Type: enumeration Possible values: `Off`, `Once`, `Continuous` GUI visibility: expert Streamable feature, stored with user configuration sets
`LvLUTBrightness` *GUI name: Brightness*	Brightness control Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets Unit: %
`LvLUTContrast` *GUI name: Contrast*	Contrast control Type: integer GUI visibility: expert Streamable feature, stored with user configuration sets Unit: %
Feature category: Transport Layer Control , refer to Section 6.4, “Transport layer control” [p. 84]	

Name	Description
`PayloadSize` *GUI name: Payload Size*	Number of bytes provided for each image by the device Type: integer GUI visibility: expert Read-only feature Unit: B
Feature category: User Sets , refer to Section 6.13, “User sets (saving/loading camera configuration)” [p. 157]	
`UserSetSelector` *GUI name: User Configuration Set Selector*	Selects configuration set to configure Type: enumeration Possible values: `Default`, `UserSet1`, `UserSet2`, `UserSet3`, `UserSet4` GUI visibility: beginner
`UserSetLoad` [`UserSetSelector`] *GUI name: User Set Load*	Loads the selected user set Type: command GUI visibility: beginner
`UserSetSave` [`UserSetSelector`] *GUI name: User Set Save*	Saves current configuration into the selected user set Type: command GUI visibility: beginner
`UserSetDefaultSelector` *GUI name: Default Startup Set*	Selects the default feature configuraiton set Type: enumeration Possible values: `Default`, `UserSet1`, `UserSet2`, `UserSet3`, `UserSet4`, `None` GUI visibility: beginner
Feature category: Chunk Data Control , refer to Section 6.11, “Chunk data” [p. 146]	
`ChunkModeActive` *GUI name: Chunk Mode Active*	Enables the chunk mode Type: bool GUI visibility: expert Streamable feature, stored with user configuration sets
`ChunkSelector` *GUI name: Chunk Selector*	Selects the chunk to configure Type: enumeration Possible values: `OffsetX`, `OffsetY`, `Width`, `Height`, `PixelFormat`, `LinePitch`, `FrameID`, `Timestamp`, `ExposureTime`, `Gain`, `LineStatusAll`, `BlackLevel`, `LvExternalADCValue`, `LvTriggerDelayed`, `LvStrobeDropped` GUI visibility: expert

Name	Description
`ChunkEnable` [`ChunkSelector`] *GUI name: Chunk Enable*	Enables the selected chunk Type: bool GUI visibility: expert Streamable feature, stored with user configuration sets
`ChunkOffsetX` *GUI name: X Offset*	X offset applied to image included in the payload Type: integer GUI visibility: expert Read-only feature Unit: px
`ChunkOffsetY` *GUI name: Y Offset*	Y offset applied to image included in the payload Type: integer GUI visibility: expert Read-only feature Unit: px
`ChunkWidth` *GUI name: Width*	Width of the image included in the payload Type: integer GUI visibility: expert Read-only feature Unit: px
`ChunkHeight` *GUI name: Height*	Height of the image included in the payload Type: integer GUI visibility: expert Read-only feature Unit: px
`ChunkPixelFormat` *GUI name: Pixel Format*	Pixel format of the image included in the payload Type: enumeration Possible values: `Mono8`, `Mono8Signed`, `Mono10`, `Mono10Packed`, `Mono12`, `Mono12Packed`, `Mono16`, `BayerGR8`, `BayerRG8`, `BayerGB8`, `BayerBG8`, `BayerGR10`, `BayerRG10`, `BayerGB10`, `BayerBG10`, `BayerGR12`, `BayerRG12`, `BayerGB12`, `BayerBG12`, `RGB8Packed`, `BGR8Packed`, `RGBA8Packed`, `BGRA8Packed` GUI visibility: expert Read-only feature
`ChunkLinePitch` *GUI name: Line Pitch*	Line pitch of the image included in the payload Type: integer GUI visibility: expert Read-only feature Unit: B

Name	Description
`ChunkFrameID` *GUI name: Frame ID*	Frame id of the image included in the payload Type: integer GUI visibility: expert Read-only feature
`ChunkTimestamp` *GUI name: Timestamp*	Timestamp associated with the image included in the payload Type: integer GUI visibility: expert Read-only feature Unit: us
`ChunkExposureTime` *GUI name: Exposure Time*	Exposure time used to acquire the image included in the payload Type: float GUI visibility: expert Read-only feature Unit: us
`ChunkGainSelector` *GUI name: Gain Selector*	Gain type to be reported in chunk data Type: enumeration Possible values: `AnalogAll`, `DigitalAll` GUI visibility: expert
`ChunkGain` [`ChunkGainSelector`] *GUI name: Gain*	Gain used to acquire the image included in the payload Type: float GUI visibility: expert Read-only feature Unit: dB
`ChunkBlackLevel` *GUI name: Black Level*	Black level used to acquire the image included in the payload Type: float GUI visibility: expert Read-only feature Unit: DN
`ChunkLineStatusAll` *GUI name: Line Status All*	Status of i/o lines at time of acquiring the image in the payload Type: integer GUI visibility: expert Read-only feature

Name	Description
`ChunkLvExternalADCSelector` *GUI name: External ADC Selector*	>External ADC to be reported in chunk data Type: enumeration Possible values: `ExternalADC1`, `ExternalADC2`, `ExternalADC3`, `ExternalADC4` GUI visibility: expert
`ChunkLvExternalADCValue` [`ChunkLvExternalADCSelector`] *GUI name: External ADC Value*	Value of the selected external ADC at time of acquisition of the image included in the payload Type: integer GUI visibility: expert Read-only feature Unit: DN
`ChunkLvTriggerDelayed` *GUI name: Trigger Delayed*	Flag indicating if the trigger was delayed when acquiring the image included in the payload Type: bool GUI visibility: expert Read-only feature
`ChunkLvStrobeDropped` *GUI name: Strobe Dropped*	Flag indicating if the configured strobe was dropped when acquiring the image included in the payload Type: bool GUI visibility: expert Read-only feature
Feature category: Event Control , refer to Section 6.12, “Asynchronous events” [p. 151]	
`EventSelector` *GUI name: Event Selector*	Selects which event to signal to the host application. Type: enumeration Possible values: `LvLog`, `LvTriggerDropped` GUI visibility: expert
`EventNotification` [`EventSelector`] *GUI name: Event Notification*	Activate or deactivate the notification to the host application of the selected event occurrence. Type: enumeration Possible values: `Off`, `On` GUI visibility: expert Streamable feature, stored with user configuration sets
`EventLvLog` *GUI name: Log Event*	Returns the unique identifier of the log type of event. Type: integer GUI visibility: expert Read-only feature

Name	Description
`EventLvLogTimestamp` *GUI name: Log Event Timestamp*	Returns the timestamp of the log event. Type: integer GUI visibility: expert Read-only feature Unit: us
`EventLvLogMessage` *GUI name: Log Event Message*	The log message coming with the event Type: string GUI visibility: expert Read-only feature
`EventLvTriggerDropped` *GUI name: Trigger Dropped Event*	Returns the unique identifier of the dropped trigger type of event. Type: integer GUI visibility: expert Read-only feature
`EventLvTriggerDroppedTimestamp` *GUI name: Trigger Dropped Event Timestamp*	Returns the timestamp of the dropped trigger event. Type: integer GUI visibility: expert Read-only feature Unit: us

Table 6.2. Summary of all camera features (some of them available only by specific models)

7. Simplon GenTL Producer features

The behaviour of the Simplon GenTL Producer can be controlled through feature trees, similarly as the features of the camera itself (the camera features were documented in Chapter 6, *Camera features and their control* [p. 79]). The basic principles of work with the Simplon GenTL Producer features are same as with the camera features, refer to the introduction in Section 6.1, "Common principles" [p. 79].

This chapter will discuss the Simplon GenTL Producer features, some of them dictated by the GenTL standard, others specific to Simplon. The chapter will discuss the Simplon GenTL Producer in general, considering all Leutron Vision camera families. Note that the chapter is irrelevant for users connecting to our GigE Vision cameras (PicSight-GigE/PicSight-Smart) directly through the GigE Vision interface of a 3rd party library — in such case the Simplon GenTL Producer is not involved.

A feature set is associated with every GenTL module instance. The module types are:

- System — represents the entire GenTL producer.
- Interface — an interface through which the cameras (devices) are connected to the system.
- Device — GenTL producer's view of the camera (device).
- Data stream — controls image acquisition from the camera.
- Buffer — wraps a single acquisition buffer. Simplon GenTL Producer currently does not provide any buffer related features.

The availability and use of the individual features is the same, regardless if you access the Simplon GenTL Producer through Simplon API or through a 3rd party GenTL compatible library. Individual libraries might slightly differ in details such as how the individual GenTL modules are enumerated and opened, but the functionality will be the same in the end. For more information about the GenTL producer access and configuration, refer to Simplon programmer's guide or to documentation of the 3rd party library you are using. You might also want to study Chapter 8, *Connecting with 3rd party software packages* [p. 215].

Overview of all the supported features is in Section 7.5, "Simplon GenTL Producer feature summary" [p. 208].

The Simplon API extends the individual feature sets with even more features for each "GenTL module". These features are, however, discussed in the corresponding manual, Simplon programmer's guide.

On CheckSight/CheckSight-II cameras, just a single instance of the Simplon GenTL Producer can be running at a moment. The first instance gains full control over the system and does not allow sharing it with other instances. This limitation does not apply on other systems, using PicSight-GigE cameras, where multiple instances can be running in parallel.

7.1. GenTL system module

The system module represents the entire GenTL producer, there's always exactly one instance of this module. It allows to obtain basic information about the GenTL producer, enumerate its available device interfaces and obtain basic information about the interfaces even before actually opening them.

Figure 7.1. Example GUI showing the GenTL system module features

7.1.1. Information about the system

The system module provides basic set of features describing the GenTL producer implementation:

Transport Layer Vendor — Name of the vendor implementing given GenTL producer. For Simplon GenTL Producer this naturally reads `Leutron Vision`.

Transport Layer Model — Name of the GenTL producer itself, in our case `Simplon GenTL Producer`.

Transport Layer ID — Unique ID of the transport layer producer. This feature is used for communication between the GenTL producer and consumer, it should not be interpreted anyhow by the user.

Transport Layer Version — String representation of the GenTL producer's version number. This feature provides the version number of the installed Simplon package.

Transport Layer Path — Path to the GenTL producer file (the `.cti` file). The exact location depends on how the Simplon package was installed.

Transport Layer Type — Another feature dictated by the GenTL specification. It informs the consumer about the camera interface technologies it supports. For Simplon GenTL Producer it's always *Mixed*, because the producer supports multiple camera interfaces (CheckSight/CheckSight-II cameras and GigE Vision compatible cameras).

GenTL Version Major/GenTL Version Minor — Major/minor version of the GenTL specification implemented by the producer.

GigE Vision Version Major/GigE Vision Version Minor — Major/minor version of the GigE Vision specification implemented by the GenTL producer's component handling GigE Vision cameras.

Simplon

```
LvHSystem hSys; // open the system to get the handle
// read basic information about the system
char StringVal[1024];
LvGetString (hSys, LvSystem_TLVendorName, StringVal, 1024);
LvGetString (hSys, LvSystem_TLModelName, StringVal, 1024);
```

```
LvGetString (hSys, LvSystem_TLID, StringVal, 1024);
LvGetString (hSys, LvSystem_TLVersion, StringVal, 1024);
LvGetString (hSys, LvSystem_TLPath, StringVal, 1024);
int32_t EnumVal;
LvGetEnum(hSys, LvSystem_TLType, &EnumVal);
int32_t IntVal;
LvGetInt32 (hSys, LvSystem_GenTLVersionMajor, &IntVal);
LvGetInt32 (hSys, LvSystem_GenTLVersionMinor, &IntVal);
LvGetInt32 (hSys, LvSystem_GevVersionMajor, &IntVal);
LvGetInt32 (hSys, LvSystem_GevVersionMinor, &IntVal);
```

GEN<i>CAM

```
// read basic information about the system
read_string (TLVendorName)
read_string (TLModelName)
read_string (TLID)
read_string (TLVersion)
read_string (TLPath)
read_enum (TLType)
read_int (GenTLVersionMajor)
read_int (GenTLVersionMinor)
read_int (GevVersionMajor)
read_int (GevVersionMinor)
```

7.1.2. Interface enumeration

The system module also provides means for GenTL interface enumeration and querying basic information about the interfaces without need to open them. Note that some of the GigE Vision specific features are only available on the GigE Vision interfaces — and even only if the producer creates one interface module for every network card (see also Section 7.2, “GenTL interface module” [p. 198]).

Update Interface List

Command asking the GenTL producer to rescan the currently available camera interfaces and update its internal interface list.

Note that Simplon API executes this command internally when enumerating/discovering the interfaces, so there's no need to access this feature directly when using Simplon API.

Interface Selector

Selects one of the discovered interfaces and allows to query information about the interfaces through the depending features (see below).

The *Interface Selector* is an integer feature with zero-based indexing of the discovered interfaces. The index corresponds with the same index in the GenTL C interface. The current range of the selector (zero through its current maximum) corresponds with the number of discovered interfaces.

Note that in Simplon API the interfaces are accessed through corresponding handles and the interface indexes/ID's are fully wrapped into the handles — they can be usually ignored by Simplon API users.

Interface ID	Unique ID of the selected interface. This feature is used for communication between the GenTL producer and consumer, it should not be interpreted anyhow by the user.
Interface MAC Address	MAC address corresponding with the selected interface. GigE Vision interfaces only.
Interface Default IP Address	Default IP address configured on the selected interface. GigE Vision interfaces only.
Interface Default Subnet Mask	Default subnet mask configured on the selected interface. GigE Vision interfaces only.
Interface Default Gateway	Default gateway configured on the selected interface. GigE Vision interfaces only.

Simplon

```
LvHSystem hSys; // open the system to get the handle
// rescan the list of available interfaces
// (waiting for operation completion)
LvCmdExecute (hSys, LvSystem_InterfaceUpdateList);
// (now wait until the operation end using LvCmdIsDone())
// read information about the first discovered interface
// (i.e. interface with index 0)
LvSetInt32 (hSys, LvSystem_InterfaceSelector, 0);
char StringVal[1024];
LvGetString (hSys, LvSystem_InterfaceID, StringVal, 1024);
if (LvIsAvailable (hSys, LvSystem_GevInterfaceMACAddress))
{
    // MAC/IP addresses can be read as integers or formatted strings
    int32_t IntVal;
    int64_t Int64Val;
    LvGetInt64 (hSys, LvSystem_GevInterfaceMACAddress, &Int64Val);
    LvGetString (hSys, LvSystem_GevInterfaceMACAddress, StringVal, 1024);
    LvGetInt32 (hSys, LvSystem_GevInterfaceDefaultIPAddress, &IntVal);
    LvGetInt32 (hSys, LvSystem_GevInterfaceDefaultSubnetMask, &IntVal);
    LvGetInt32 (hSys, LvSystem_GevInterfaceDefaultGateway, &IntVal);
}
```

GEN<i>CAM

```
// rescan the list of available interfaces
// (waiting for operation completion)
exec_cmd (InterfaceUpdateList)
wait_until_done (InterfaceUpdateList)
// read information about the first discovered interface
// (i.e. interface with index 0)
write_int (InterfaceSelector, 0)
read_string (InterfaceID)
read_int (GevInterfaceMACAddress)
read_int (GevInterfaceDefaultIPAddress)
```

```
read_int (GevInterfaceDefaultSubnetMask)
read_int (GevInterfaceDefaultGateway)
```

7.2. GenTL interface module

The interface module symbolizes a physical interface through which the GenTL producer discovers and connects to the cameras (“devices”). It allows to obtain basic information about the interface, enumerate the devices currently accessible through the interface and obtain basic information about the devices even before actually opening them.

- The CheckSight/CheckSight-II cameras are always connected through a “PCI Interface”. The name indicates that the camera's acquisition module is connected to the system through PCI-Express.
- The GigE Vision cameras (PicSight-GigE, PicSight-Smart or any other GigE Vision compatible cameras) connect through dedicated interface(s). Depending on Simplon version and/or configuration there might be either a single common logical interface called “GigE Interface” which encapsulates all physical network interfaces (network cards) available in the system. Alternately, there might be one GenTL interface per physical network card. In the latter case, the network related information features (such as MAC address) are reported. When using the single logical interface, representing all the network cards, the per-card features are hidden.

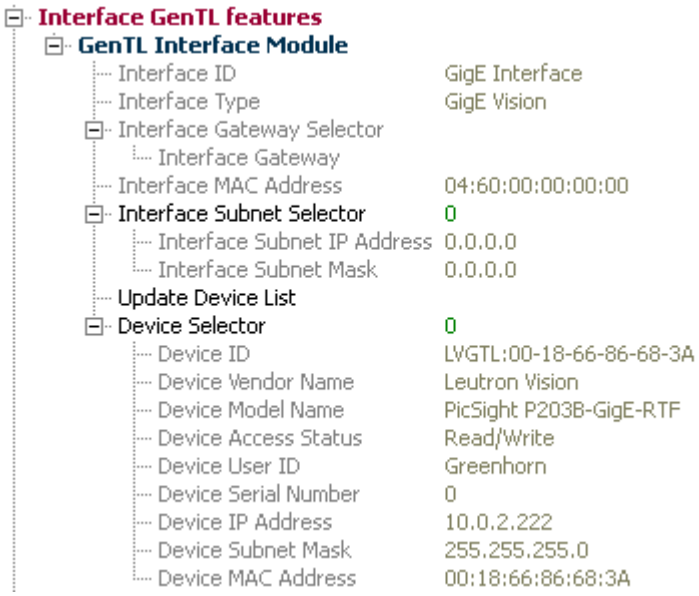


Figure 7.2. Example GUI showing the GenTL interface module features

7.2.1. Information about the interface

The interface module provides basic set of features describing its properties:

Interface ID — Unique ID of the interface. This feature is used for communication between the GenTL producer and consumer, it should not be interpreted anyhow by the user.

Interface Type — A feature dictated by the GenTL specification. It informs the consumer about the camera interface technology it supports. For Simplon GenTL Producer it's always *GigE Vision* for the interface(s) handling the GigE Vision cameras (such as PicSight-GigE) and *Custom* for the interface controlling CheckSight/CheckSight-II.

Interface MAC Address	MAC address corresponding with the interface. GigE Vision interfaces only.
Interface gateways' information	The *Interface Gateway Selector* feature (zero based integer index) allows to loop through the gateways configured for given GigE Vision interface. The *Interface Gateway* then reports the IP address of the selected gateway. GigE Vision interfaces only.
Interface subnets' information	The *Interface Subnet Selector* feature (zero based integer index) allows to loop through the subnets configured for given GigE Vision interface. The *Interface Subnet IP Address*/*Interface Subnet Mask* then report the IP address and subnet mask corresponding with the selected subnet. GigE Vision interfaces only.

Simplon

```
LvHInterface hItf; // open the interface to get the handle
// read basic information about the interface
char StringVal[1024];
LvGetString (hItf, LvInterface_InterfaceID, StringVal, 1024);
int32_t EnumVal;
LvGetEnum(hItf, LvInterface_InterfaceType, &EnumVal);
if (LvIsAvailable (hItf, LvInterface_GevMACAddress))
{
    // MAC/IP addresses can be read as integers or formatted strings
    int32_t IntVal;
    int64_t Int64Val;
    LvGetInt64 (hItf, LvInterface_GevMACAddress, &Int64Val);
    LvGetString (hItf, LvInterface_GevMACAddress, StringVal, 1024);
    // first gateway
    LvSetInt32 (hItf, LvInterface_GevInterfaceGatewaySelector, 0);
    LvGetInt32 (hItf, LvInterface_GevInterfaceGateway, &IntVal);
    // first subnet
    LvSetInt32 (hItf, LvInterface_GevInterfaceSubnetSelector, 0);
    LvGetInt32 (hItf, LvInterface_GevInterfaceSubnetIPAddress, &IntVal);
    LvGetInt32 (hItf, LvInterface_GevInterfaceSubnetMask, &IntVal);
}
```

GEN<i>CAM

```
// read basic information about the interface
read_string (InterfaceID)
read_enum (InterfaceType)
read_int (GevMACAddress)
// first gateway
write_int (GevInterfaceGatewaySelector, 0)
read_int (GevInterfaceGateway)
// first subnet
write_int (GevInterfaceSubnetSelector, 0)
read_int (GevInterfaceSubnetIPAddress)
read_int (GevInterfaceSubnetMask)
```

7.2.2. Device enumeration

The interface module also provides means for device (camera) enumeration and querying basic information about the devices without need to open them. Note that some of the GigE Vision specific features are only available for the GigE Vision devices Unlike with interface enumeration, the device enumeration might be done repeatedly at runtime by some applications since devices (in particular the GigE Vision cameras) might go away or get connected to the system any time, so the device list is more dynamic by its nature.

Update Device List

Command asking the GenTL producer to rescan the list of devices currently accessible through given interface.

Keep in mind that after the update, the contents of the list of the devices “known” to the Simplon GenTL Producer might change, including their order. Handles to the opened devices are guaranteed to remain valid, however.

Note that Simplon API executes this command internally when enumerating/discovering the devices, so there's no need to access this feature directly when using Simplon API.

Device Selector

Selects one of the discovered devices and allows to query information about the devices through the depending features (see below).

The *Device Selector* is an integer feature with zero-based indexing of the discovered devices. The index corresponds with the same index in the GenTL C interface. The current range of the selector (zero through its current maximum) corresponds with the number of discovered devices.

Note that in Simplon API the devices are accessed through corresponding handles and the device indexes/ID's are fully wrapped into the handles — they can be usually ignored by Simplon API users.

Device ID

Globally unique ID of the selected device. This feature is used for communication between the GenTL producer and consumer, it should not be interpreted anyhow by the user.

Device Vendor Name

Name of the selected camera's vendor (manufacturer).

Device Model Name

Model name of the selected camera.

Device Access Status

Access status of the selected device at the time of the last device list update. This enumeration feature can report following status options: *Read/Write* (the device can be opened for reading and writing), *Read Only* (the device can be opened in read-only mode), *No Access* (the device cannot be currently opened — e.g. because it's opened by another application already).

Device User ID

User ID (also known as nickname) of the selected device. The user ID's can be assigned by the user to individual cameras — and used as custom device identification scheme. In Simplon API the User ID can be directly used as device (camera) identifier when opening it.

Device Serial Number

Serial number (as a string) of the selected device.

Device MAC Address

MAC address of the selected device.

GigE Vision devices only.

Device IP Address	Current IP address of the selected device. GigE Vision devices only.
Device Subnet Mask	Current subnet mask of the selected device. GigE Vision devices only.

Simplon

```
LvHInterface hItf; // open the interface to get the handle
// rescan the list of available devices
// (waiting for operation completion)
LvCmdExecute (hItf, LvInterface_DeviceUpdateList);
// (now wait until the operation end using LvCmdIsDone())
// read information about the first discovered device
// (i.e. device with index 0)
LvSetInt32 (hItf, LvInterface_DeviceSelector, 0);
char StringVal[1024];
LvGetString (hItf, LvInterface_DeviceID, StringVal, 1024);
LvGetString (hItf, LvInterface_DeviceVendorName, StringVal, 1024);
LvGetString (hItf, LvInterface_DeviceModelName, StringVal, 1024);
LvGetString (hItf, LvInterface_LvDeviceUserID, StringVal, 1024);
LvGetString (hItf, LvInterface_LvDeviceSerialNumber, StringVal, 1024);
int32_t EnumVal;
LvGetEnum(hItf, LvInterface_DeviceAccessStatus, &EnumVal);
// MAC/IP addresses can be read as integers or formatted strings
int32_t IntVal;
int64_t Int64Val;
LvGetInt64 (hItf, LvInterface_GevDeviceMACAddress, &Int64Val);
LvGetString (hItf, LvInterface_GevDeviceMACAddress, StringVal, 1024);
LvGetInt32 (hItf, LvInterface_GevDeviceIPAddress, &IntVal);
LvGetInt32 (hItf, LvInterface_GevDeviceSubnetMask, &IntVal);
```

GEN<i>CAM

```
// rescan the list of available devices
// (waiting for operation completion)
exec_cmd (DeviceUpdateList)
wait_until_done (DeviceUpdateList)
// read information about the first discovered device
// (i.e. device with index 0)
write_int (DeviceSelector, 0)
read_string (DeviceID)
read_string (DeviceVendorName)
read_string (DeviceModelName)
read_string (LvDeviceUserID)
read_string (LvDeviceSerialNumber)
read_enum (DeviceAccessStatus)
read_int (GevDeviceMACAddress)
read_int (GevDeviceIPAddress)
read_int (GevDeviceSubnetMask)
```

7.3. GenTL device module

The device module provides a GenTL producer's local view of each opened camera (“device”) — there's one instance for every device you open. Note that this does not refer to the features exported by the camera itself (those were discussed in Chapter 6, *Camera features and their control* [p. 79]), but rather to the features exported by the GenTL producer itself, controlling how it handles the camera.

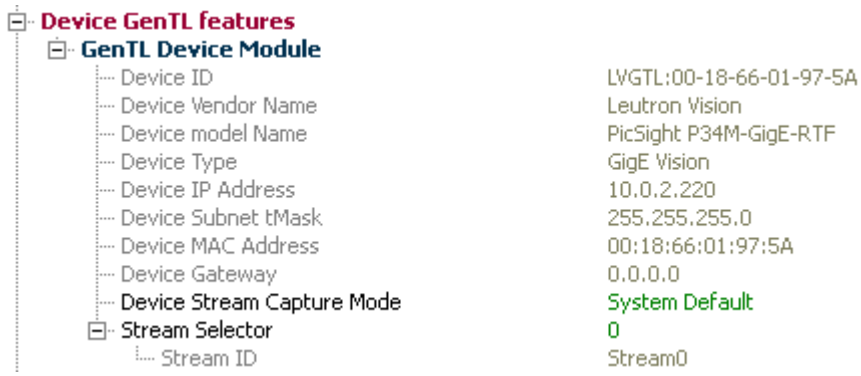


Figure 7.3. Example GUI showing the GenTL device module features

7.3.1. Information about the device

The device module provides basic set of features describing its properties:

Device ID — Globally unique ID of the device. This feature is used for communication between the GenTL producer and consumer, it should not be interpreted anyhow by the user.

Device Type — A feature dictated by the GenTL specification. It informs the consumer about the camera's interface technology it is based on. For Simplon GenTL Producer it's always *GigE Vision* for the GigE Vision cameras (such as PicSight-GigE) and *Custom* for the CheckSight/CheckSight-II cameras.

Device Vendor Name — Name of the camera's vendor (manufacturer).

Device Model Name — Model name of the camera.

Device MAC Address — MAC address of the device.

GigE Vision devices only.

Device IP Address — Current IP address of the device.

GigE Vision devices only.

Device Subnet Mask — Current subnet mask of the device.

GigE Vision devices only.

Device Gateway — Current default gateway of the device.

GigE Vision devices only.

```
LvHDevice hDev; // open the device to get the handle
// read basic information about the device
char StringVal[1024];
LvGetString (hDev, LvDevice_DeviceID, StringVal, 1024);
int32_t EnumVal;
LvGetEnum(hDev, LvDevice_DeviceType, &EnumVal);
LvGetString (hDev, LvDevice_DeviceVendorName, StringVal, 1024);
LvGetString (hDev, LvDevice_DeviceModelName, StringVal, 1024);
// MAC/IP addresses can be read as integers or formatted strings
int32_t IntVal;
int64_t Int64Val;
LvGetInt64 (hDev, LvDevice_GevDeviceMACAddress, &Int64Val);
LvGetString (hDev, LvDevice_GevDeviceMACAddress, StringVal, 1024);
LvGetInt32 (hDev, LvDevice_GevDeviceIPAddress, &IntVal);
LvGetInt32 (hDev, LvDevice_GevDeviceSubnetMask, &IntVal);
LvGetInt32 (hDev, LvDevice_GevDeviceGateway, &IntVal);
```

GEN<i>CAM

```
// read basic information about the device
read_string (DeviceID)
read_enum (DeviceType)
read_string (DeviceVendorName)
read_string (DeviceModelName)
read_int (GevDeviceMACAddress)
read_int (GevDeviceIPAddress)
read_int (GevDeviceSubnetMask)
read_int (GevDeviceGateway)
```

7.3.2. Stream enumeration

The interface module also provides means for stream “enumeration”, according to the GenTL specification. However, Simplon GenTL Producer currently always uses just a single stream per device. For streams, there's no “update stream list” command like with other module types, the stream (s) configuration is always known upon opening the device module and remains fixed.

Stream Selector

Selects one of the streams belonging to the device and allows to query information about the stream through the depending features (see below).

The *Stream Selector* is an integer feature with zero-based indexing of the device streams. The index corresponds with the same index in the GenTL C interface. The current range of the selector (zero through its current maximum) corresponds with the number of streams of the device.

Note that in Simplon API the streams are accessed through corresponding handles and the stream indexes/ID's are fully wrapped into the handles — they can be usually ignored by Simplon API users.

Stream ID

Unique ID of the selected stream. This feature is used for communication between the GenTL producer and consumer, it should not be interpreted anyhow by the user.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// read information about the first stream of the device
// (i.e. stream with index 0)
```

```
LvSetInt32 (hDev, LvDevice_StreamSelector, 0);
char StringVal[1024];
LvGetString (hDev, LvDevice_StreamID, StringVal, 1024);
```

GEN<i>CAM

```
// read information about the first stream of the device
// (i.e. stream with index 0)
write_int (StreamSelector, 0)
read_string (StreamID)
```

7.3.3. GigE Vision streaming control

This topic is specific to GigE Vision devices only. When acquiring images from a GigE Vision camera, depending on the camera's data rate and configuration, the amount of data being sent over the network in the image stream can be high enough to generate significant processor load and consume other system resources — especially when being processed through the regular operating system's network stack. The Simplon GenTL Producer provides a network filter driver to minimize such processing overhead.

The feature *Device Stream Capture Mode* allows to select, which stream capture mode should be used for given device. Possible options are:

- *System Default* (the default option) — the capture mode will be selected according to the default option adjusted in the Simplon configuration file (Section 5.3, “Simplon configuration” [p. 76]). Name of the corresponding configuration (boolean) option is `Default use of filter driver`.
- *Socket* — the image data from this camera will be captured through the regular socket interface, regardless of the global default configuration.
- *Filter Driver* — the image data from this camera will be captured through the filter driver interface, regardless of the global default configuration.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// use system default GigE Vision stream capture mode
// (configurable through Simplon ini-file)
LvSetEnum (hDev, LvDevice_LvGevDeviceStreamCaptureMode,
                 LvGevDeviceStreamCaptureMode_SystemDefault);
// capture the GigE Vision image stream through standard socket
LvSetEnum (hDev, LvDevice_LvGevDeviceStreamCaptureMode,
                 LvGevDeviceStreamCaptureMode_Socket);
// capture the GigE Vision image stream through the filter driver
LvSetEnum (hDev, LvDevice_LvGevDeviceStreamCaptureMode,
                 LvGevDeviceStreamCaptureMode_FilterDriver);
```

GEN<i>CAM

```
// use system default GigE Vision stream capture mode
// (configurable through Simplon ini-file)
write_enum (LvGevDeviceStreamCaptureMode, SystemDefault)
// capture the GigE Vision image stream through standard socket
write_enum (LvGevDeviceStreamCaptureMode, Socket)
// capture the GigE Vision image stream through the filter driver
write_enum (LvGevDeviceStreamCaptureMode, FilterDriver)
```

7.3.4. GigE Vision communication control

This topic is specific to GigE Vision devices only. When controlling a GigE Vision camera, the Simplon GenTL Producer (and every GigE Vision compliant software) uses the GVCP protocol, which basically consists of commands issued by the host software and acknowledges responded by the device. If the acknowledge from the device does not arrive within the expected timeout, Simplon GenTL Producer assumes that delivery of the command failed and eventually retries sending it again. Both the timeout and number of eventual retries can be configured by the user through features *Control Channel Transmission Timeout* (value in ms) and *Control Channel Retry Count*.

If the device itself provides hints about the maximal expectable response timeout, Simplon GenTL Producer uses this hint to adjust the initial value of *Control Channel Transmission Timeout*, otherwise it defaults to 200 ms.

The default values used by Simplon GenTL Producer will work just fine in vast majority of cases and the user should attempt to readjust them only if encountering a communication problem (such as failing read/write operations) with a particular camera, usually with guidance of our support staff. The features become visible only at the “guru” feature visibility level. Note that it is important to configure these values before the problematic communication with the device can ever happen, ie. typically before accessing the feature tree of the remote device.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// adjust control channel retransmission parameters
LvSetInt32 (hDev, LvDevice_LvGevCCRC, 3);
LvSetInt32 (hDev, LvDevice_LvGevCCTT, 500); // ms
```

GENICAM

```
// adjust message channel retransmission parameters
write_int (LvGevCCRC, 3)
write_int (LvGevCCTT, 500) // ms
```

7.3.5. Determining the GigE Vision streaming packet size

This topic is specific to GigE Vision devices only. When acquiring the image data from a GigE Vision camera, it is usually desirable to stream the data using larger packets to improve the performance. The possible packet size value is, however, limited by the capabilities of the networking hardware involved. When connecting to a camera, the Simplon GenTL Producer determines the maximal packet size usable with the current network configuration and adjusts it on the camera. For applications needing to further control the packet size while working with the camera, Simplon GenTL Producer provides two additional features (available only in the “guru” feature visibility level). Remember, that changing the streaming packet size on the fly might be tricky and these features must never be used while any streaming is active.

When the application suspects that the conditions affecting the maximal possible packet size are incorrect or if the packet size became otherwise invalid, the procedure to determine the maximal possible packet size can be executed again using the *Find Maximal Packet Size* command. When the operation finishes, the resulting maximal packet size usable with the device and current network configuration can be retrieved from *Packet Size Value*. The application can then actually apply the packet size on the camera itself (using the Packet Size™ feature in the remote device feature tree) whenever suitable — but not during active streaming.

For applications requiring even finer approach to packet size control, the Simplon GenTL Producer can be asked to verify if a given packet size is valid on the current network configuration. The specified packet size would be verified, but not applied — this is left for the application. The

packet size to be tested should be specified in *Packet Size Value* and the verification procedure started using the *Test Packet Size* command. The success (usability of given packet size) is reported in *Packet Size Test Success*.

Simplon

```
LvHDevice hDev; // open the device to get the handle
// determine the maximal usable packet size...
LvCmdExecute (hDev, LvDevice_LvGevFindMaxPacketSize);
// ...now wait until the operation end using LvCmdIsDone()
// and read the result (will not be automatically applied,
// the application has to configure the camera explicitly)
int32_t IntVal;
LvGetInt32 (hDev, LvDevice_LvGevPacketSizeValue, &IntVal);
// test if packet size 9000 is usable with current configuration...
LvSetInt32 (hDev, LvDevice_LvGevPacketSizeValue, 9000);
LvCmdExecute (hDev, LvDevice_LvGevTestPacketSize);
// ...now wait until the operation end using LvCmdIsDone()
// and read the result
LvGetBool (hDev, LvDevice_LvGevPacketSizeTestSuccess, &IntVal);
```

GEN<i>CAM

```
// determine the maximal usable packet size...
exec_cmd (LvGevFindMaxPacketSize)
// ...now wait until the operation end using command's IsDone property
// (will not be automatically applied,
// the application has to configure the camera explicitly)
read_int (LvGevPacketSizeValue)
// test if packet size 9000 is usable with current configuration...
write_int (LvGevPacketSizeValue, 9000)
exec_cmd (LvGevTestPacketSize)
// ...now wait until the operation end using command's IsDone property
// and read the result
read_bool (LvGevPacketSizeTestSuccess)
```

7.4. GenTL data stream module

The stream module controls the image acquisition from the device. In the moment, the Simplon GenTL Producer is always using just a single data stream per device.

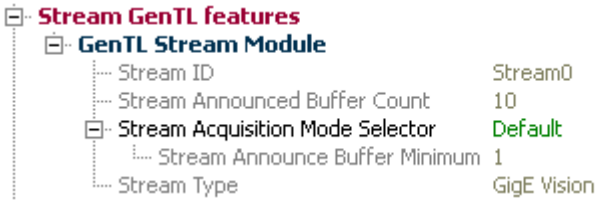


Figure 7.4. Example GUI showing the GenTL data stream module features

7.4.1. Information about the stream

The data stream module provides basic set of features describing its properties:

Stream ID
Unique ID of the stream. This feature is used for communication between the GenTL producer and consumer, it should not be interpreted anyhow by the user.

Stream Type
A feature dictated by the GenTL specification. It informs the consumer about the camera interface technology of the camera device the stream belongs to. For Simplon GenTL Producer it's always *GigE Vision* for streams belonging to the GigE Vision cameras (such as PicSight-GigE) and *Custom* for stream belonging to CheckSight/CheckSight-II.

Stream Announced Buffer Count
Number of buffers currently announced by the GenTL consumer to the stream. This value is "volatile", it can be changed at any time by the consumer when adding/removing acquisition buffers from the stream.

Stream Acquisition Mode Selector
Selects the acquisition mode to be used by the stream. Acquisition mode describes how the data stream handles the pool of input acquisition buffers, which buffers it delivers first to the consumer (application) and how buffer locking behaves.

Simplon GenTL Producer currently supports just a single acquisition mode — the default acquisition mode required by the GenTL specification. In this mode the buffers are delivered to the consumer (application) in FIFO order, the application must explicitly queue (unlock) all the buffers after processing to make them again available to the acquisition engine and the acquisition engine never overwrites any buffers in the output queue — when overflown, new images are dropped.

Stream Announce Buffer Minimum
Minimum number of acquisition buffers the stream requires for operation. The value depends on the acquisition mode. For the current version of Simplon GenTL Producer and the default acquisition mode the minimum is `1`.

Simplon

```
LvHStream hStrm; // open the stream to get the handle
// read basic information about the stream
char StringVal[1024];
LvGetString (hStrm, LvStream_StreamID, StringVal, 1024);
int32_t EnumVal;
LvGetEnum(hStrm, LvStream_StreamType, &EnumVal);
int32_t IntVal;
LvGetInt32 (hStrm, LvStream_StreamAnnouncedBufferCount, &IntVal);
LvGetEnum(hStrm, LvStream_StreamAcquisitionModeSelector, &EnumVal);
LvGetInt32 (hStrm, LvStream_StreamAnnounceBufferMinimum, &IntVal);
```

GEN<i>CAM

```
// read basic information about the stream
read_string (StreamID)
read_enum (StreamType)
read_int (StreamAnnouncedBufferCount)
read_enum (StreamAcquisitionModeSelector)
read_int (StreamAnnounceBufferMinimum)
```

7.5. Simplon GenTL Producer feature summary

The following table provides condensed summary of all the features provided by the Simplon GenTL Producer for individual GenTL modules. Some of the features are only relevant for certain camera families. The range of allowed values might differ depending on the actual system configuration.

For detailed explanation of the purpose of each feature group and their dependencies, refer to previous sections in this chapter.

Name	Description
Feature category: GenTL System Module , refer to Section 7.1, “GenTL system module” [p. 194]	
`TLVendorName` *GUI name: Transport Layer Vendor*	GenTL producer vendor name Type: string GUI visibility: beginner Read-only feature
`TLModelName` *GUI name: Transport Layer Model*	GenTL producer model name Type: string GUI visibility: beginner Read-only feature
`TLID` *GUI name: Transport Layer ID*	GenTL producer ID Type: string GUI visibility: beginner Read-only feature
`TLVersion` *GUI name: Transport Layer Version*	GenTL producer version Type: string GUI visibility: beginner Read-only feature
`TLPath` *GUI name: Transport Layer Path*	GenTL producer path Type: string GUI visibility: beginner Read-only feature
`TLType` *GUI name: Transport Layer Type*	GenTL producer type (currently always reported as 'Mixed') Type: enumeration Possible values: `Mixed`, `Custom`, `GEV` GUI visibility: beginner Read-only feature
`GenTLVersionMajor` *GUI name: GenTL Version Major*	Major version of the implemented GenTL specification Type: integer GUI visibility: beginner Read-only feature

Name	Description
`GenTLVersionMinor` *GUI name: GenTL Version Minor*	Minor version of the implemented GenTL specification Type: integer GUI visibility: beginner Read-only feature
`GevVersionMajor` *GUI name: GigE Vision Version Major*	Major version of the implemented GigE Vision specification Type: integer GUI visibility: beginner Read-only feature
`GevVersionMinor` *GUI name: GigE Vision Version Minor*	Minor version of the implemented GigE Vision specification Type: integer GUI visibility: beginner Read-only feature
`InterfaceUpdateList` *GUI name: Update Interface List*	Update internal list of interfaces Type: command GUI visibility: beginner
`InterfaceSelector` *GUI name: Interface Selector*	Selects an interface module to be accessed Type: integer GUI visibility: beginner
`InterfaceID` [`InterfaceSelector`] *GUI name: Interface ID*	Interface ID Type: string GUI visibility: beginner Read-only feature
`GevInterfaceMACAddress` [`InterfaceSelector`] *GUI name: Interface MAC Address*	MAC address of the (GigE Vision) interface Type: integer GUI visibility: beginner Read-only feature
`GevInterfaceDefaultIPAddress` [`InterfaceSelector`] *GUI name: Interface Default IP Address*	Default IP address of the (GigE Vision) interface Type: integer GUI visibility: beginner Read-only feature
`GevInterfaceDefaultSubnetMask` [`InterfaceSelector`] *GUI name: Interface Default Subnet Mask*	Default subnet mask of the (GigE Vision) interface Type: integer GUI visibility: beginner Read-only feature

Name	Description
`GevInterfaceDefaultGateway` [`InterfaceSelector`] *GUI name: Interface Default Gateway*	Default gateway of the (GigE Vision) interface Type: integer GUI visibility: beginner Read-only feature
Feature category: GenTL Interface Module , refer to Section 7.2, “GenTL interface module” [p. 198]	
`InterfaceID` *GUI name: Interface ID*	Interface ID Type: string GUI visibility: beginner Read-only feature
`InterfaceType` *GUI name: Interface Type*	Interface type Type: enumeration Possible values: `Custom`, `GEV` GUI visibility: beginner Read-only feature
`GevInterfaceGatewaySelector` *GUI name: Interface Gateway Selector*	Selects gateway associated with a (GigE Vision) interface Type: integer GUI visibility: beginner
`GevInterfaceGateway` [`GevInterfaceGatewaySelector`] *GUI name: Interface Gateway*	Gateway of a (GigE Vision) interface Type: integer GUI visibility: beginner Read-only feature
`GevMACAddress` *GUI name: Interface MAC Address*	MAC address of a (GigE Vision) interface Type: integer GUI visibility: beginner Read-only feature
`GevInterfaceSubnetSelector` *GUI name: Interface Subnet Selector*	Selects subnet associated with a (GigE Vision) interface Type: integer GUI visibility: beginner
`GevInterfaceSubnetIPAddress` [`GevInterfaceSubnetSelector`] *GUI name: Interface Subnet IP Address*	Subnet IP address of a (GigE Vision) interface Type: integer GUI visibility: beginner Read-only feature
`GevInterfaceSubnetMask` [`GevInterfaceSubnetSelector`] *GUI name: Interface Subnet Mask*	Subnet mask of a (GigE Vision) interface Type: integer GUI visibility: beginner Read-only feature

Name	Description
`DeviceUpdateList` *GUI name: Update Device List*	Update internal list of devices Type: command GUI visibility: beginner
`DeviceSelector` *GUI name: Device Selector*	Selects a device module to be accessed Type: integer GUI visibility: beginner
`DeviceID` [`DeviceSelector`] *GUI name: Device ID*	Device ID Type: string GUI visibility: beginner Read-only feature
`DeviceVendorName` [`DeviceSelector`] *GUI name: Device Vendor Name*	Device vendor name Type: string GUI visibility: beginner Read-only feature
`DeviceModelName` [`DeviceSelector`] *GUI name: Device Model Name*	Device model name Type: string GUI visibility: beginner Read-only feature
`DeviceAccessStatus` [`DeviceSelector`] *GUI name: Device Access Status*	Access status of the device at time of last device list update Type: enumeration Possible values: `ReadWrite`, `ReadOnly`, `NoAccess` GUI visibility: beginner Read-only feature
`LvDeviceUserID` [`DeviceSelector`] *GUI name: Device User ID*	Device User ID Type: string GUI visibility: beginner Read-only feature
`LvDeviceSerialNumber` [`DeviceSelector`] *GUI name: Device Serial Number*	Device identifier (serial number) Type: string GUI visibility: beginner Read-only feature
`GevDeviceIPAddress` [`DeviceSelector`] *GUI name: Device IP Address*	IP address of the (GigE Vision) device Type: integer GUI visibility: beginner Read-only feature
`GevDeviceSubnetMask` [`DeviceSelector`] *GUI name: Device Subnet Mask*	Subnet mask of the (GigE Vision) device Type: integer GUI visibility: beginner Read-only feature

Name	Description
`GevDeviceMACAddress` [`DeviceSelector`] *GUI name: Device MAC Address*	MAC address of the (GigE Vision) device Type: integer GUI visibility: beginner Read-only feature
Feature category: GenTL Device Module , refer to Section 7.3, "GenTL device module" [p. 202]	
`DeviceID` *GUI name: Device ID*	Device ID Type: string GUI visibility: beginner Read-only feature
`DeviceVendorName` *GUI name: Device Vendor Name*	Device vendor name Type: string GUI visibility: beginner Read-only feature
`DeviceModelName` *GUI name: Device Model Name*	Device model name Type: string GUI visibility: beginner Read-only feature
`DeviceType` *GUI name: Device Type*	Device type Type: enumeration Possible values: `Custom`, `GEV` GUI visibility: beginner Read-only feature
`GevDeviceIPAddress` *GUI name: Device IP Address*	IP address of the (GigE Vision) device Type: integer GUI visibility: beginner Read-only feature
`GevDeviceSubnetMask` *GUI name: Device Subnet Mask*	Subnet mask of the (GigE Vision) device Type: integer GUI visibility: beginner Read-only feature
`GevDeviceMACAddress` *GUI name: Device MAC Address*	MAC address of the (GigE Vision) device Type: integer GUI visibility: beginner Read-only feature
`GevDeviceGateway` *GUI name: Device Gateway*	Gateway of the (GigE Vision) device Type: integer GUI visibility: beginner Read-only feature

Name	Description
`DeviceEndianessMechanism` *GUI name: Device Endianess Mechanism*	Identifies the endianess mode Type: enumeration Possible values: `Legacy`, `Standard` GUI visibility: guru
`LvGevDeviceStreamCaptureMode` *GUI name: Device Stream Capture Mode*	Device stream capture mode Type: enumeration Possible values: `SystemDefault`, `Socket`, `FilterDriver` GUI visibility: expert Streamable feature, stored with user configuration sets
`LvGevFindMaxPacketSize` *GUI name: Find Maximal Packet Size*	Determine the maximal usable packet size for streaming Type: command GUI visibility: guru
`LvGevPacketSizeValue` *GUI name: Packet Size Value*	Streaming packet size to be verified using the Test Packet Size command Type: integer GUI visibility: guru
`LvGevTestPacketSize` *GUI name: Test Packet Size*	Test if the selected packet size is suitable for streaming in current network configuration Type: command GUI visibility: guru
`LvGevPacketSizeTestSuccess` *GUI name: Packet Size Test Success*	Reports success of the last packet size test command Type: bool GUI visibility: guru Read-only feature
`LvGevCCTT` *GUI name: Control Channel Transmission Timeout*	Controls the GigE Vision control channel transmission timeout value in milliseconds. Type: integer GUI visibility: guru Unit: ms
`LvGevCCRC` *GUI name: Control Channel Retry Count*	Controls the number of GigE Vision control channel command retransmissions allowed when a control channel command times out. Type: integer GUI visibility: guru
`StreamSelector` *GUI name: Stream Selector*	Selects a stream module to be accessed Type: integer GUI visibility: beginner

Name	Description
`StreamID` [`StreamSelector`] *GUI name: Stream ID*	Stream ID Type: string GUI visibility: beginner Read-only feature
Feature category: GenTL Stream Module , refer to Section 7.4, "GenTL data stream module" [p. 206]	
`StreamID` *GUI name: Stream ID*	Stream ID Type: string GUI visibility: beginner Read-only feature
`StreamAnnouncedBufferCount` *GUI name: Stream Announced Buffer Count*	Number of buffers announced for the data stream Type: integer GUI visibility: beginner Read-only feature
`StreamAcquisitionModeSelector` *GUI name: Stream Acquisition Mode Selector*	Selects desired acquisition mode Type: enumeration Possible values: `Default` GUI visibility: beginner
`StreamAnnounceBufferMinimum` [`StreamAcquisitionModeSelector`] *GUI name: Stream Announce Buffer Minimum*	Minimum number of buffers to be announced for selected acquisition mode Type: integer GUI visibility: beginner Read-only feature
`StreamType` *GUI name: Stream Type*	Stream type Type: enumeration Possible values: `Custom`, `GEV` GUI visibility: beginner Read-only feature
Feature category: GenTL Buffer Module	
The GenTL producer currently does not provide any features for this module.	

Table 7.1. Summary of all GenTL producer features (some of them available only for specific camera technologies)

8. Connecting with 3rd party software packages

Since the basic principles of integrating individual camera models in the 3rd party products are frequently same for all the camera families, this chapter discusses all the product lines.

Our cameras connect to all GenTL compliant libraries, PicSight-GigE/PicSight-Smart cameras then also to all GigE Vision compliant libraries. The libraries include

- Halcon and ActivVisionTools from MVTec Software GmbH (GenTL and GigE Vision compliant)
- OpenCV, open source library originally developed by Intel (Simplon Source Code Generator provides samples to connect with OpenCV)
- LabView from National Instruments (GigE Vision compliant)
- VisionPro from Cognex (GigE Vision compliant)
- Common Vision Blox from Stemmer Imaging (GenTL and GigE Vision compliant)
- MIL (Matrox Imaging Library) from Matrox (GigE Vision compliant)
- MATLAB from MathWorks (GenTL and GigE Vision compliant)
- Adaptive Vision from Future Processing (Future Processing provides connectivity to PicSight-Smart cameras)
- Any other GenTL or GigE Vision compliant software.

The following sections provide basic instructions and some useful hints about using Leutron Vision cameras and Simplon GenTL Producer with some of the major 3rd party software packages. If you don't find instructions for the software library of your choice, then simply follow the official documentation of that software package or contact our support team in case of experiencing a problem.

8.1. Halcon (MVTec Software GmbH)

This section provides hints for successful integration of our cameras in the Halcon library (and ActivVisionTools). The details of the Halcon image acquisition interface is subject to change by MVTec Software GmbH. In case of any discrepancies, the official MVTec Software GmbH documentation overrides the info in this manual. If in doubts, please contact MVTec Software GmbH to get more accurate information.

Some of the points below might be targeted for use in the HDevelop environment. It should be well usable, however, for all Halcon users.

8.1.1. GigE Vision interface (PicSight-GigE, PicSight-Smart)

When working with our GigE Vision cameras, the “GigEVision” image acquisition interface of Halcon should be used. This interface accesses the GigE Vision cameras directly, no Leutron Vision software needs to be installed. During the initial tests, it might still be good idea to install the Simplon software package for easy camera configuration or possible troubleshooting.

If desired, however, the cameras might also be connected through the “GenICamTL” acquisition interface, as described in the next section.

- Be sure to always use the latest version of the GigEVision acquisition interface released for your Halcon version. It may contain important updates. Read its documentation carefully.
- To discover the GigE Vision cameras connected to the system, use the `info_framegrabber` operator:

  ```
  info_framegrabber ('GigEVision', 'info_boards', Information, ValueList)
  ```

  That call will rescan the network and detect all currently available GigE Vision cameras, providing various information about every detected camera. The most important part of the string is so called “unique name”, which is built as *`MacAddress_VendorName_ModelName`*, where any

non-alphanumeric characters are eliminated from vendor and model names. Example unique name can be `00186601975a_LeutronVision_PicSightP34MGigE`. The unique name is used when opening the camera in the `open_framegrabber` operator, passing the unique name in the `Device` parameter. Alternatively, the entire output string from the `info_framegrabber` can be used to open the camera as well.

The `info_framegrabber` operator has an alternative form:

```
info_framegrabber ('GigEVision', 'device', Information, ValueList)
```

In this case it returns just the unique names, without the additional information.

To make the discussion complete, the cameras can be also opened using the *Device User ID* string (sometimes also known as camera “nickname”). Once you assign unique user ID's to all used cameras, referring them using those ID's might be easier and more straightforward than using the Halcon “unique names”. Of course, this mechanism can only work reliably, if the user ID's for individual cameras are truly unique (no duplicities).

When you need to rescan the system to discover newly connected cameras, just call the `info_framegrabber` operator again.

- The PicSight-GigE/PicSight-Smart cameras might not be properly discovered if their IP configuration does not match the network interface they are connected to, for example if they are configured for different subnet. In such case, it's recommended to use the Simplon GigE Vision Configuration Tool to configure all the cameras first, before using them in Halcon.

8.1.2. GenTL interface (CheckSight/CheckSight-II, optionally PicSight-GigE/PicSight-Smart)

When working with our CheckSight/CheckSight-II cameras, the “GenICamTL” image acquisition interface of Halcon should be used. This interface accesses the cameras through Simplon GenTL Producer, according to the GenICam GenTL standard. Therefore the Simplon package has to be installed in the system.

Halcon will find and use the Simplon GenTL Producer automatically. The GenTL producers are (according to the standard) registered in the system through the `GENICAM_GENTL32_PATH`/`GENICAM_GENTL64_PATH` environment variable. This is done automatically by the Simplon installer. However, on some Linux systems not supporting the `/etc/profile.d` mechanism, this might fail. In such case, you will need to set the environment variable manually according to the conventions used in your Linux distribution. The variable should point to the directory containing the Simplon GenTL Producer, `liblv.gentl.cti`. This directory is by default `/opt/simplon/cti` on 32-bit, `/opt/simplon/cti64` on 64-bit systems.

- Be sure to always use the latest version of the GenICamTL acquisition interface released for your Halcon version. It may contain important updates. Read its documentation carefully.
- To discover the cameras connected to the system, use the `info_framegrabber` operator:

```
info_framegrabber ('GenICamTL', 'info_boards', Information, ValueList)
```

That call will rescan the system and detect all cameras currently visible through the “GenTL producers” installed in the system, providing various information about every detected camera. The three most important components of the information string are the GenTL “device ID” of the camera, the ID of the “GenTL interface” the camera was discovered on and finally the corresponding GenTL producer itself (identified by its full path). These three components are enough to open the camera using the `open_framegrabber` operator. The string passed to the operator in the `Device` parameter should contain these three components, using similar syntax as returned from `info_framegrabber`, eg. `device:DeviceID`

`interface:`*`InterfaceID`* `producer:`*`ProducerPath`*. Alternatively, the entire output string from the `info_framegrabber` can be used to open the camera as well.

The `info_framegrabber` operator has an alternative form:

```
info_framegrabber ('GigEVision', 'device', Information, ValueList)
```

In this case it returns just the GenTL device ID's, without the additional information. The device ID unequivocally identifies the camera, so it can be also used to open it. However, this mechanism currently only works in Halcon when the camera is detected on the first interface of the first GenTL producer, therefore it's not fully reliable and we recommend to use the full syntax (device/interface/producer).

When you need to rescan the system to discover newly connected cameras, just call the `info_framegrabber` operator again. Note that Halcon currently avoids rescanning the GenTL interfaces where one ore more cameras are already open. This might improve in future.

- When opening the camera, you can adjust the number of acquisition buffers the Halcon acquisition interface will allocate for the camera. In most cases the default value (`3`) will be just fine, but in some scenarios it might be useful to allocate more buffers to prevent acquisition overflows, eg. when the system gets temporarily under a heavy load and cannot process the incoming images as they are coming. Larger number of acquisition buffers can help to overcome short peaks of the processing load.

  The number of buffers might be changed through the `Generic` parameter of the `open_framegrabber` operator. Set this parameter to `num_buffers=`*`N`*, where *`N`* is the desired number of buffers.

- It is possible to control also the features of the individual GenTL modules corresponding with the open camera: the Device module, its parent Interface module and the System module representing the GenTL producer itself. In such case, the module name in square brackets should prefix the feature name: `[Device]`*`FeatureName`*, `[Interface]`*`FeatureName`*, `[System]`*`FeatureName`*.

  Adjusting the parameters of the Stream and Buffer GenTL modules is not supported by the Halcon GenTL interface.

8.1.3. Common notes

- During the development phase, it might be good idea to switch on the low-level error reporting mechanism of Halcon, it can help for troubleshooting.

  ```
  set_system ('do_low_error', 'true')
  ```

  Note however, that in some situations, such as when switching on the chunk data mode or when using the variable payload size functionality, the “low-error” mechanism of Halcon might trigger a warning that the payload size (ie. size of the image data) reported by the camera is bigger than the size calculated by Halcon. Please ignore this warning, the payload size reported by the camera is always correct — the size increase is legitimate in certain operating modes.

- All integer-based device features are internally handled as signed, with native size on the current platform. This can bring certain limitations, in particular on 32-bit systems. Whenever working with integers that might exceed the 32-bit range or that are by nature unsigned, keep this in mind. The features include for example:
  - Timestamps and related features.
  - MAC and IP addresses.
  - 32-bit masks and bitfields, such as *Line Status All, User Output Value All* or similar.
  - Generic unsigned integer features used by PicSight-Smart applications.

To input an integer in hexadecimal form, use the `0x` prefix:

```
Result := 0xFF
```

To get a value displayed in hexadecimal form, use the `$'x'` appendix with the variable:

```
HexValue := Value$'x'
```

- When trying to set an integer or float value which is out of the range supported by the camera, the Halcon acquisition interface will not raise an error, but round the value to the closest supported one.
- Chunk data nor event data are currently supported by Halcon.
- Halcon image acquisition interfaces make significant efforts to allow changing any camera feature value at any time, including the features that would be “locked” (read-only) by the camera during the acquisition. When the application attempts to write to a feature that is currently read-only, Halcon stops the acquisition to check whether the camera would unlock the feature and eventually writes to it. Then it restarts the acquisition.

  This attitude brings a good deal of flexibility, but it also comes for a price. Starting and stopping the acquisition generates certain overhead on all cameras — in particular with the DMA based devices such as CheckSight/CheckSight-II. In such case the acquisition buffers must be locked into the physical memory when starting the acquisition, and that is potentially time consuming operation, depending on the timing requirements of the application. Furthermore, after changing certain parameters, the size of the data acquired for each image (“payload size”) can change — then the Halcon interface will need to reallocate all the acquisition buffers, which yields additional overhead. All of these operations might of course also fail at any time, depending on the actual status of the system resources — all this in the middle of acquisition.

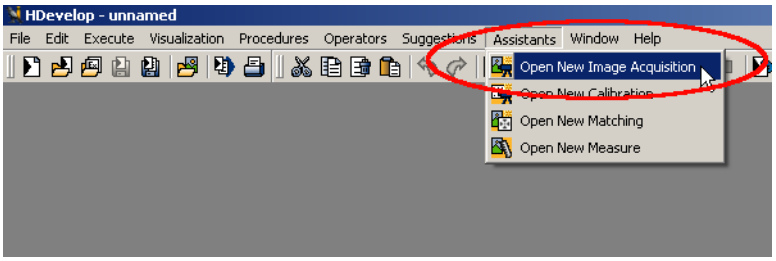
  For all these reasons we recommend to configure the features normally locked during the acquisition only at the beginning and don't touch them again during the acquisition — unless really necessary or unless the timing requirements of the application are not too strict. When not sure if a given feature is locked during the acquisition, the easiest way is to verify in the Simplon Explorer — after opening the camera and starting the acquisition, part of the features in the feature tree will become “greyed”, signalling that they were locked by the camera.

8.1.4. Halcon Image Acquisition Assistant

When starting with camera integration in Halcon, you can either start with one of the samples delivered with Halcon or use the Image Acquisition Assistant (tool built in the Halcon HDevelop development environment) to create the application skeleton. When finished with tuning the HDevelop script, the program can be exported to programming languague of your choice.

In the few steps below, we'll briefly demonstrate how to use the Image Acquisition Assistant. Note that this section does not substitute the Halcon documentation, it's just a brief introduction with some useful hints.

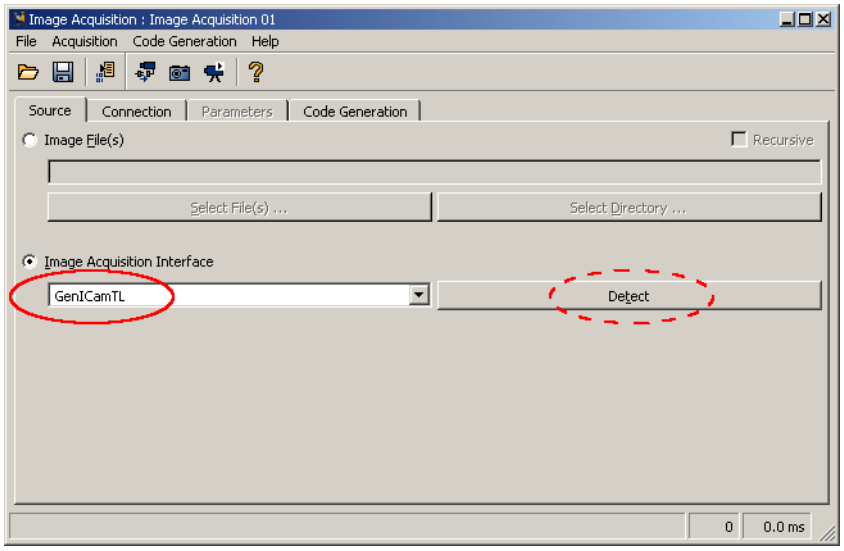
1. Start HDevelop and open a new Image Acquisition Assistant from the menu Assistants → Open New Image Acquisition.


HDevelop - unnamed
File Edit Execute Visualization Procedures Operators Suggestions Assistants Window Help
Open New Image Acquisition
Open New Calibration
Open New Matching
Open New Measure

2. Select the Halcon image acquisition interface you want to use to control the camera(s). Depending on the camera type(s), select either “GenICamTL” or “GigEVision” interface.

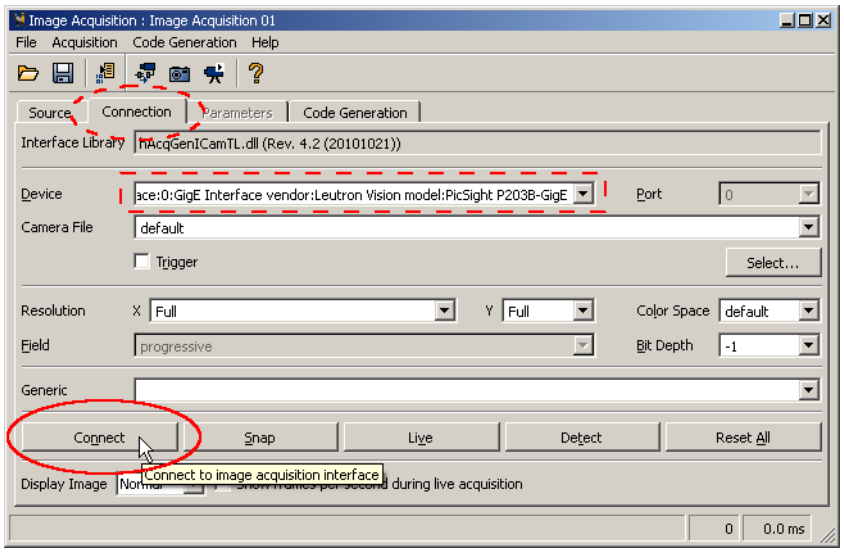
   The assistant will query the list of cameras available through that interface and allow to connect one of them at the Connection tab.

   It's possible to rescan the list of connected cameras later by clicking the Detect button. Note however, that current implementation of the Halcon GenICamTL interface does not attempt to discover cameras connected to interface on which one ore more cameras are already open. As a consequence, after opening a camera on a GenTL interface, you won't be able to start a new Image Acquisition Assistant and open another camera on the same interface — the assistant woudl not discover it. This behaviour might improve in a future version of Halcon GenICamTL acquisition interface.



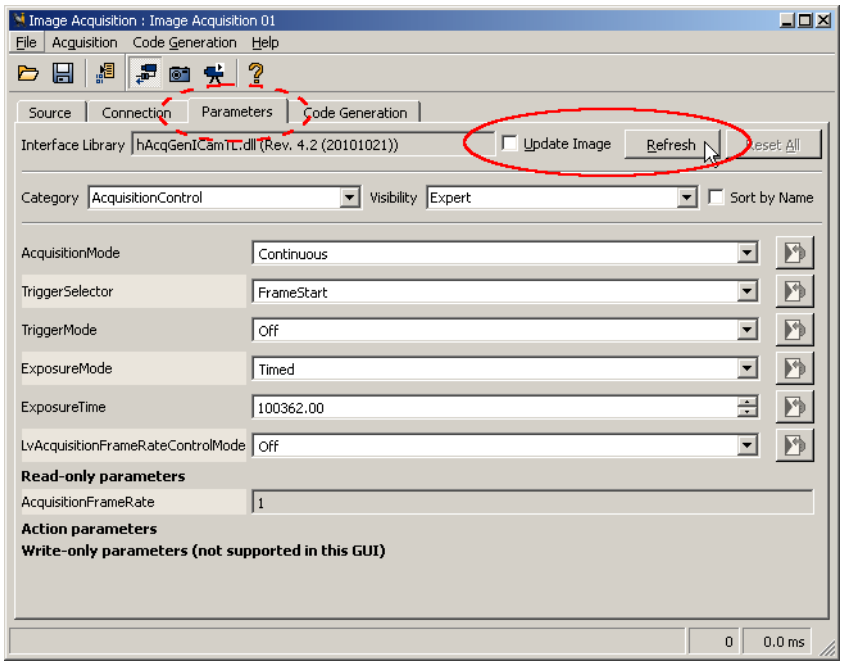
3. Switch from the Source to the Connection tab of the acquisition assistant. By the top of the window, you'll see the Device combobox listing the discovered cameras.

   Select the one you wish to work with and click the Connect button. The assistant will open the connection to the camera, retrieve it's feature tree and allow to control the features or acquire images.



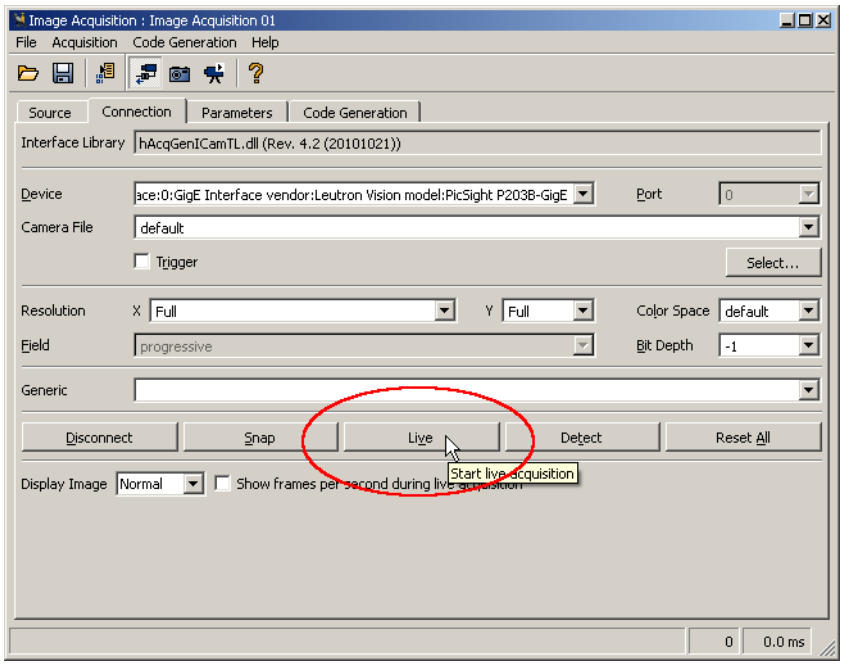
4. Switch to the Parameters tab of the assistant. Here you can fully configure the camera, the GenTL producer and even the acquisition related parameters of the Halcon itself. Some important hints:

   - The features come in three visibility levels, beginner-expert-guru. You can control them through the Visibility combobox. At the beginner level, only the basic features will be available, at the guru level all the features (including the typically “don't touch” ones) will appear. We recommend to work at the expert level in most cases.
   - Using the Category combobox the feature list can be narrowed down to a single feature category. This might make the orientation a lot easier.
   - We recommend to clear the Update Image checkbox while configuring the camera. When enabled, the assistant will attempt to acquire an image after every feature change (which frequently is not required at all). This brings an overhead in repeated acquisition start/stop loops. Indirectly, starting/stopping the acquisition might change access mode of some features (certain features are locked during acquisition). Finally, in triggered mode the acquisition won't work anyway unless you supply the trigger.
   - Halcon Image Acquisition Assistant does not handle the feature dependencies automatically. For example if changing one feature enables/disables another feature, it will not automatically appear/hide in the feature tree. To update the feature list (after changing the configuration or whenever in doubts), click the Refresh button. The feature tree will be reloaded with proper access modes.

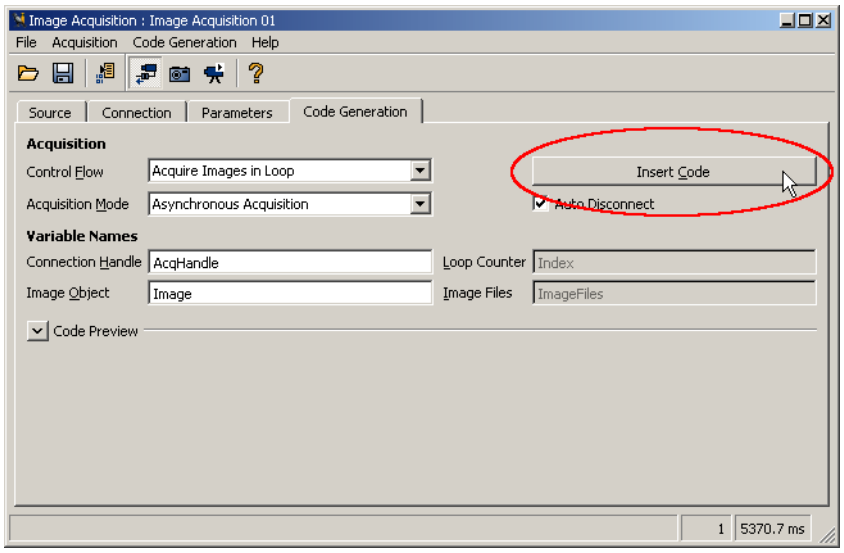


5. When finished with the camera configuration, you can test the image acquisition using the Live or Snap buttons in the Connection tab of the assistant.

   You can repeat changing the camera configuration until the desired final status is reached.



6. When finished configuring the camera, you can let the assistant to generate the application code for you. Proceed to the Code Generation tab and click the Insert Code button. The assistant will generate code that finds and opens the selected camera, adjusts all the camera parameters you have changed in the assistant and then start a simple acquisition loop. By default it will automatically disconnect from the camera so that you can immediately start playing with the generated script. The script can serve as a very good starting point for the actual application.



8.2. Common Vision Blox (Stemmer Imaging)

This section provides hints for successful integration of our cameras in the Common Vision Blox library. The details of the Common Vision Blox image acquisition interface is subject to change by Stemmer Imaging. In case of any discrepancies, the official Stemmer Imaging documentation overrides the info in this manual. If in doubts, please contact Stemmer Imaging to get more accurate information.

8.2.1. GigE Vision interface (PicSight-GigE, PicSight-Smart)

When working with our GigE Vision cameras, the Common Vision Blox image acquisition interface can access the GigE Vision cameras directly, no Leutron Vision software needs to be installed. During the initial tests, it might still be good idea to install the Simplon software package for easy camera configuration or possible troubleshooting.

If desired, however, the cameras might also be connected through the Simplon GenTL Producer, as described in the next section.

Common Vision Blox is internally using the GenTL to access the image acquisition interfaces, including its own ones (similarly as Simplon), so the functionality and user experience will be practically the same, regardless if connecting through Simplon GenTL Producer or the Common Vision Blox internal GigE Vision interface.

- Be sure to always use the latest version of the Common Vision Blox, including any available service packs (“hotfixes”). They may contain important updates. Read its documentation carefully.
- The PicSight-GigE/PicSight-Smart cameras might not be properly discovered if their IP configuration does not match the network interface they are connected to, for example if they are configured for different subnet. In such case, it's recommended to use the Simplon GigE Vision Configuration Tool (or a corresponding Common Vision Blox utility) to configure all the cameras first, before using them in Common Vision Blox.
- Chunk data are not supported by Common Vision Blox.
- Event data are supported. However, the Common Vision Blox is using a custom event delivery mechanism, not compatible with other GenTL producers. Therefore the event data delivery will work reliably only over the Common Vision Blox internal GigE Vision interface.

Common Vision Blox is expected to align to the GenTL event delivery mechanism in the future, you might wish to observe the new available updates.

8.2.2. GenTL interface (CheckSight/CheckSight-II, optionally PicSight-GigE/PicSight-Smart)

When working with our CheckSight/CheckSight-II cameras, the Common Vision Blox has to access the camera through the Simplon GenTL Producer according to the GenICam GenTL standard. Therefore the Simplon package has to be installed in the system.

Common Vision Blox is available for Windows operating systems. The Linux version is available only for the 64-bit platform, while the CheckSight/CheckSight-II is 32-bit. Therefore Linux Common Vision Blox cannot be currently used on CheckSight/CheckSight-II. If interested in the Linux operating system, please contact Stemmer Imaging to get the up to date information about plans to support 32-bit Linux version.

Common Vision Blox will find and use the Simplon GenTL Producer automatically. The GenTL producers are (according to the standard) registered in the system through the `GENICAM_GENTL32_PATH/GENICAM_GENTL64_PATH` *environment variable. This is done automatically by the Simplon installer. However, on some Linux systems not supporting the* `/etc/profile.d` *mechanism, this might fail. In such case, you will need to set the environment variable manually according to the conventions used in your Linux distribution. The variable should point to the directory containing the Simplon GenTL Producer,* `liblv.gentl.cti`*. This directory is by default* `/opt/simplon/cti` *on 32-bit,* `/opt/simplon/cti64` *on 64-bit systems.*

- Be sure to always use the latest version of the Common Vision Blox, including any available service packs (“hotfixes”). They may contain important updates. Read its documentation carefully.
- Chunk data are not supported by Common Vision Blox.
- Event data are supported. However, the Common Vision Blox is using a custom event delivery mechanism, not compatible with other GenTL producers. Therefore we strongly recommend to not use any device event related functionality within Common Vision Blox, it might cause serious problems.

  Common Vision Blox is expected to align to the GenTL event delivery mechanism in the future, you might wish to observe the new available updates.

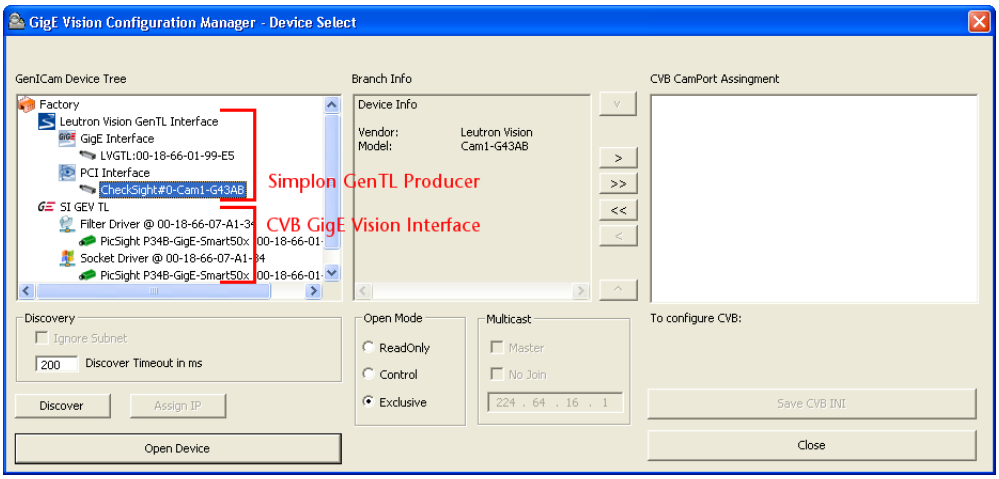
8.2.3. Getting started quickly

The Common Vision Blox comes with documentation describing how to work with the cameras connecting through the “GenICam Driver” — the Common Vision Blox component responsible for communication over the GenTL interface. That driver will be used to control our cameras, either through Simplon GenTL Producer or through Common Vision Blox own GigE Vision interface.

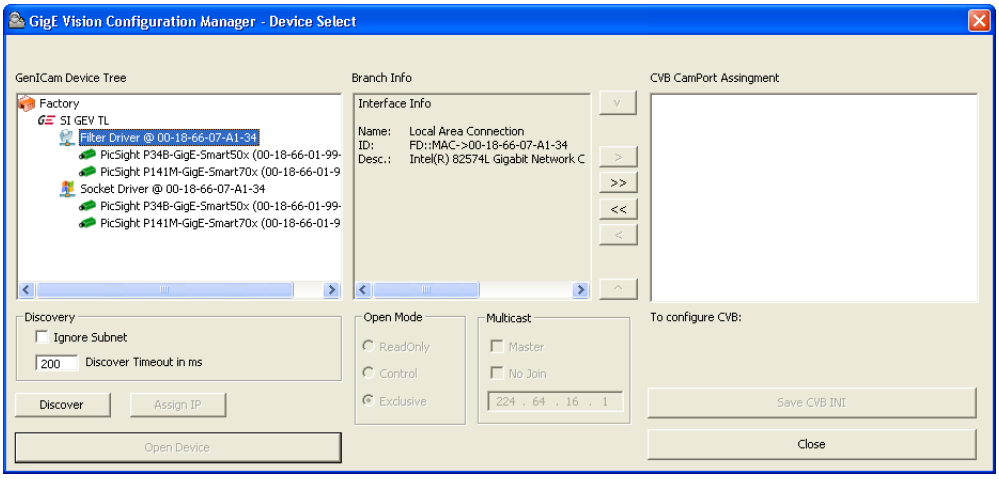
This section is not intended to replace the Common Vision Blox documentation, but rather provide step-by-step introduction on how to test and integrate our cameras into the Common Vision Blox environment. We still recommend to read the Common Vision Blox documentation carefully.

For the first test you can use the tool called GigE Vision Configuration Manager. Despite it's name, the tool is not working just with GigE Vision cameras. The executable path is `%CVB%\Hardware\StemmerImaging\siFilterDriver\GEVConfigManager.exe`, where `%CVB%` stands for Common Vision Blox installation directory.

When loaded, the tool displays the list of cameras discovered in the system. It will show cameras discovered through the Simplon GenTL Producer (both PicSight and CheckSight cameras) and through the Common Vision Blox native GigE Vision interface. Note that the PicSight cameras will be discovered multiple times, through both Simplon and Common Vision Blox interfaces.

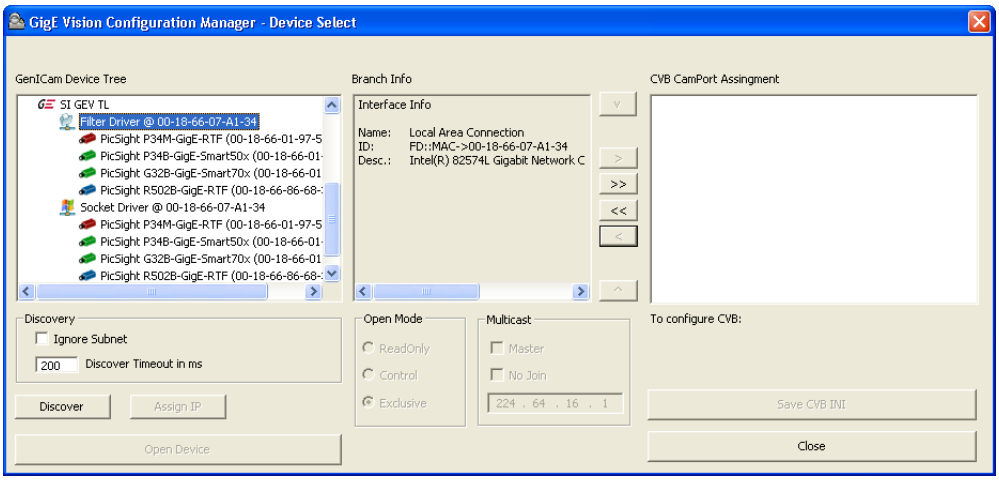


When the Simplon package is not installed, only the cameras discovered through the Common Vision Blox native acquisition interface will be recognized. Since it is GigE Vision compliant, it will detect any PicSight-GigE and PicSight-Smart cameras. You can select whether to communicate with the camera through the Common Vision Blox filter driver or through a simple socket interface.

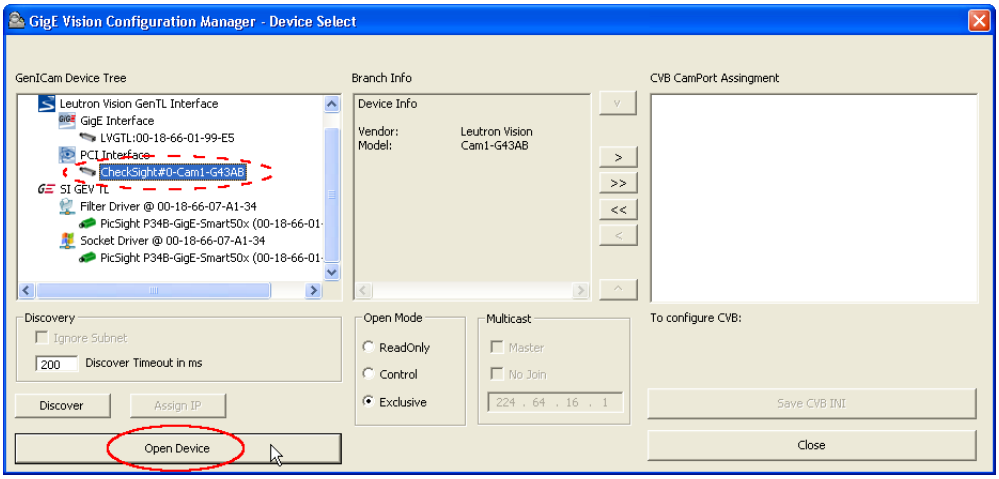


The current camera status is expressed through the camera icons:

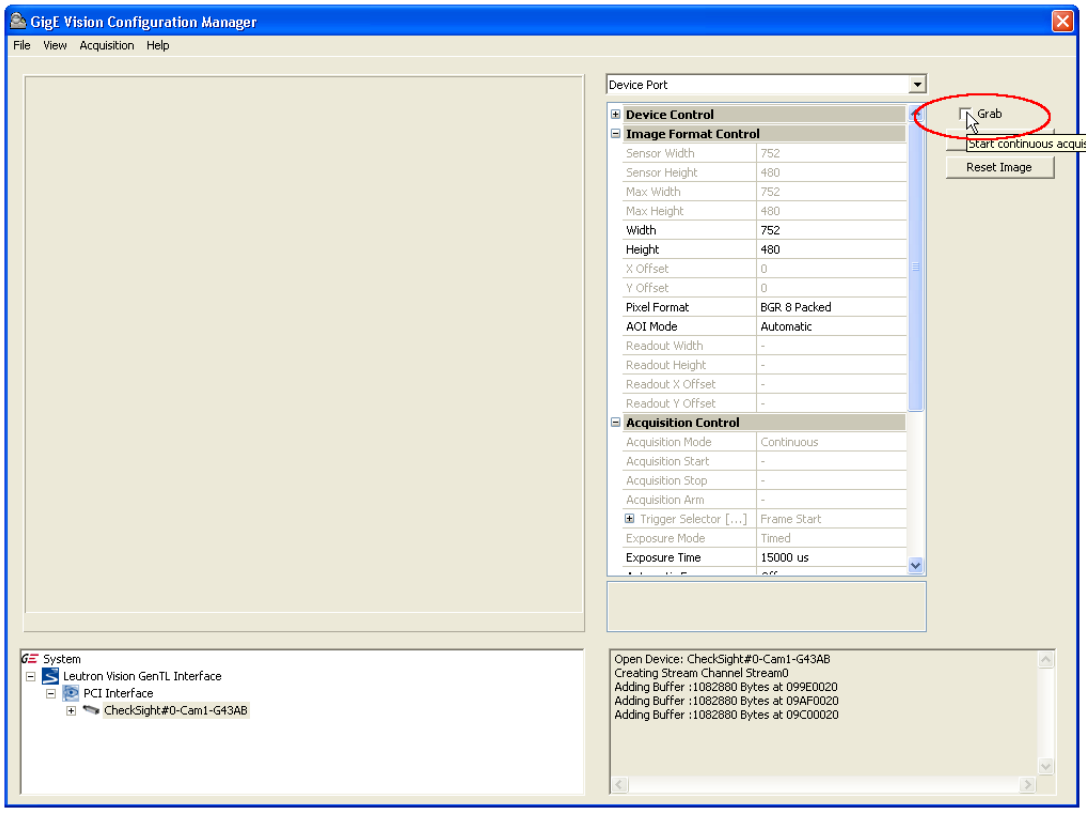
- Green icon — the camera is ready for connection.
- Blue icon — the camera is accessible, but not available in the moment, usually because it is already opened in another application (possible even from another computer).
- Red icon — the camera is not accessible. For GigE Vision cameras this usually means that the camera is currently configured for a different subnet than the computer itself, therefore they cannot communicate directly.



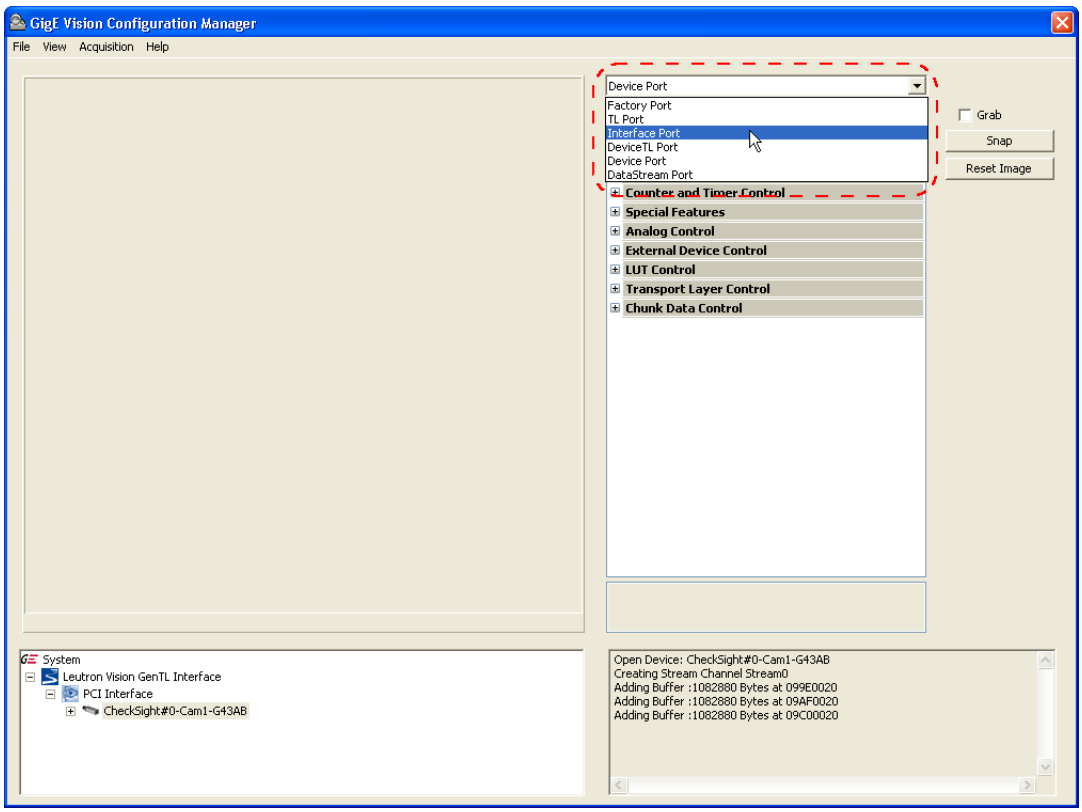
To connect to the camera, simply select it in the list and click the Open Device button.



The tool will connect to the camera and display it's feature tree. Now you can configure the desired camera parameters — and when finished, start acquisition by clicking the Grab checkbox. Clicking it again will stop the acquisition.



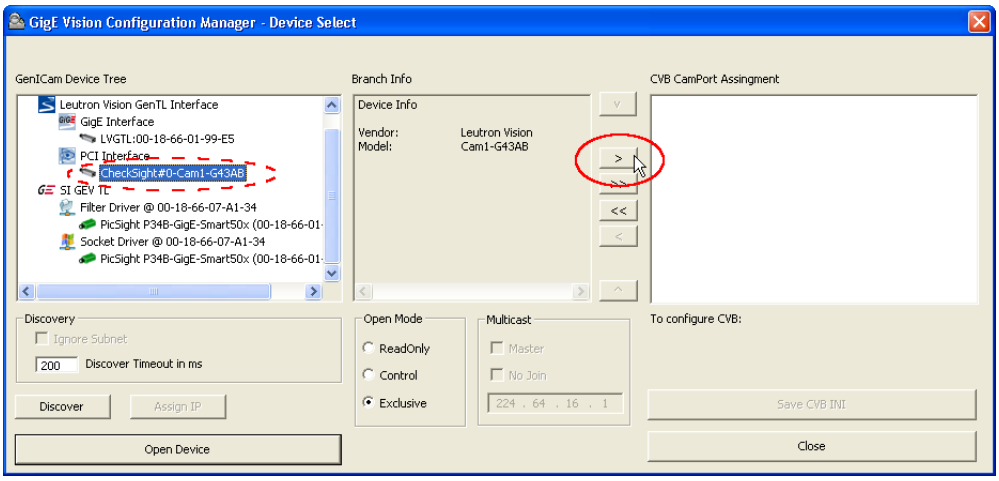
Except of the feature tree of the camera itself, feature trees of the individual components of the GenTL producer controlling the camera can be also accessed. The individual feature trees are selectable through the combobox by the top of the window.



When finished with testing the camera functionality in the GigE Vision Configuration Manager, you can use the tool to configure the camera(s) for use through the Common Vision BloxGenICam Driver. Note that the Common Vision Blox libraries and tools will access the cameras exactly through this driver.

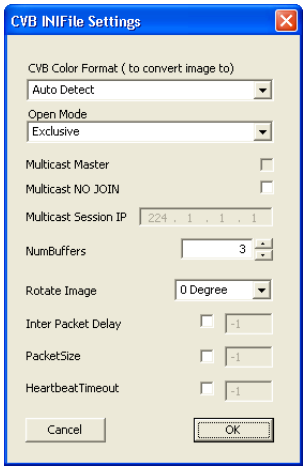
The Common Vision Blox acquisition drivers are stored in `%CVB%\Drivers` directory. The GenICam Driver binary file name is `GenICam.vin` and its configuration file `GenICam.ini`. The configuration file mainly lists the cameras to be used and their connection parameters. It can be created from the GigE Vision Configuration Manager.

Select the camera(s) you want to store in the driver's configuration file and click the > to copy them from the GenICam Device Tree panel to CVB CamPort Assignment.

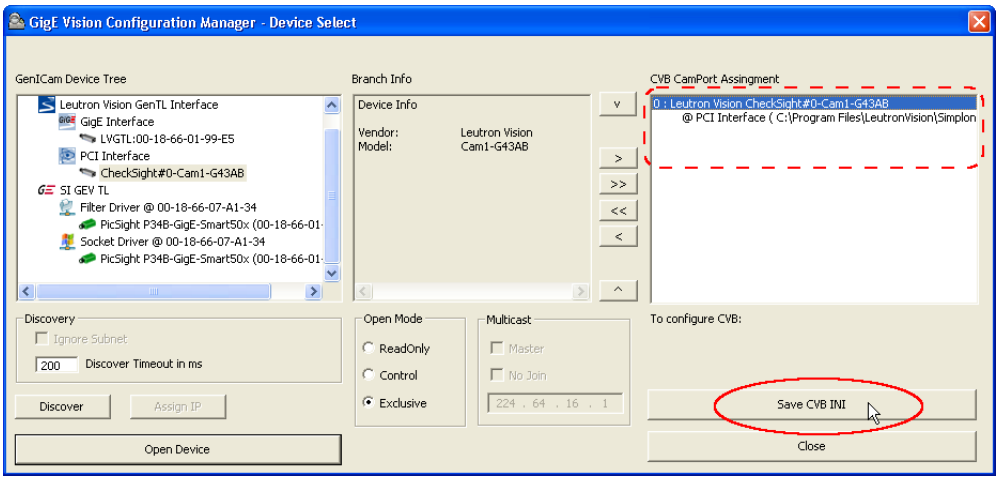


Double-clicking the camera entry in the CVB CamPort Assignment panel pops a dialog allowing to configure some parameters controlling how the driver will handle the camera.

Part of the parameters are GigE Vision specific (packet size, inter-packet delay, heartbeat timeout). Other parameters are universal, such as the `NumBuffers` parameter. It specifies how many acquisition buffers should the driver allocate for the camera. Note that when your application generates a variable processing load, it might get behind the acquisition rate in processing peaks. Increasing the number of acquisition buffers can help to overcome these peaks and help to prevent loosing images. Note also that when the driver will do color format conversion, that number of buffers will be actually allocated twice, one set for the actually acquired data, the other for the processed ones.

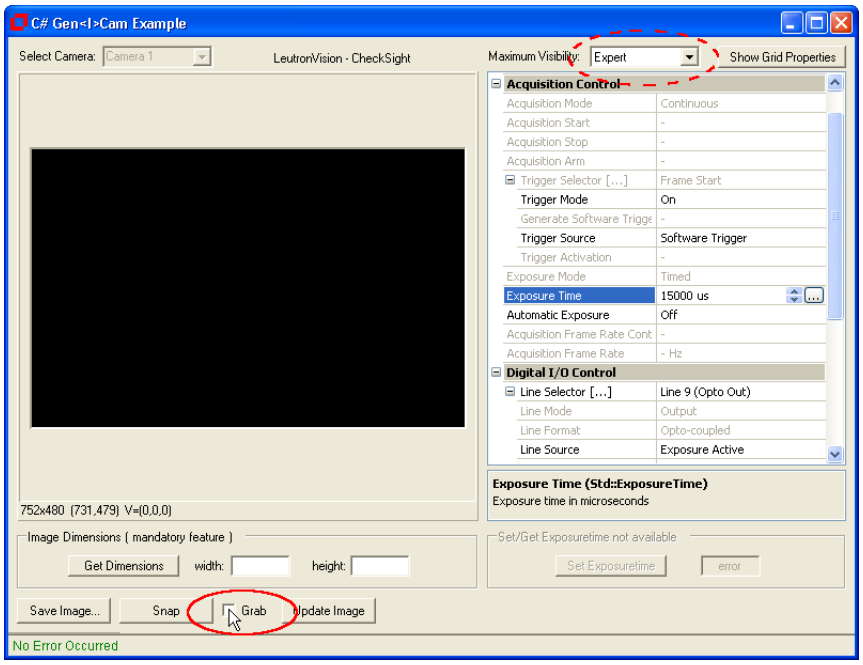


When the configuration is finished, it can be written to the driver's configuration file by clicking the Save CVB INI button.



The configuration can be verified for example through one of the Common Vision Blox sample programs. As an example, let's take the “C# GenICam Example”. Start it through system menu Start → Programs → STEMMER IMAGING → Common Vision Blox → Hardware → GenICam → C# GenICam Example.

When started, the sample will automatically open the camera configured in the driver's configuration file — provided that it is still available. It will allow to control all the camera features and then start the camera acquisition — again through the Grab checkbox.



At this point, the system should be verified and configured for use in a real Common Vision Blox application. The details how to access the cameras programmatically through the Common Vision Blox acquisition interface are beyond scope of this manual. Please refer to the official Common Vision Blox documentation.

8.3. VisionPro, CVL (Cognex)

The *Simplon Imaging Device Adapter for Cognex Software* allows to access the Leutron Vision hardware from Cognex software (VisionPro 5.2 or later or CVL 6.6 or later), through the Simplon library. This adapter is provided namely as a solution for running Cognex software on CheckSight; for

Leutron Vision GigE cameras can be used the GigE-Vision protocol, which is directly supported by Cognex software and thus this adapter is not needed.

8.3.1. Installation

The adapter is installed by Simplon Setup from version 1.00.074. It consists of 2 files:

- `lv.cognex.adapter.dll`, which is installed to the `Simplon\Bin` folder of Simplon
- `lv.cognex.adapter.ini`, which is installed to the Simplon user folder (on Windows XP is this folder the same as `Simplon\Bin`).

The Setup also modifies the Windows Registry to make the adapter visible to Cognex software. This includes the following 4 values added to 32-bit registry:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Cognex\AIK\AdapterLeutronVision]
"UseExe"=dword:00000001
"ServerNameBase"="Cognex.AIKserver.AdapterLeutronVision."
"PoolSize"=dword:08000000
"LibName"="C:\\Prg\\Simplon\\Bin\\lv.cognex.adapter.dll"
```

Note that the `PoolSize` is officially recommended to have a value `0x4000000`, however, when working with large color images (1600x1200 and larger), this size may not be sufficient and you can experience a crash (it is a known bug in Cognex software). Thus the Setup sets the value `0x8000000` instead.

Important note: When you install Cognex software on CheckSight, be sure you do not accidentally install the support for the FG8600 frame grabber. To be sure, please verify after the Cognex software installation that there is no FG8600.sys driver file installed in Windows\System32\Drivers directory. Also be sure you do not have installed the Leutron Vision LV-SDS. These drivers are in conflict with Simplon and can cause system hang.

8.3.2. Functionality

The Adapter provides the functionality required by the Cognex *Acquisition Integration Kit*. As this API is quite simple, some additional parameters, not covered by the API, can be specified in the `lv.cognex.adapter.ini` file. The contents of the file is as follows:

```
[Interfaces]
ListGigEDevices=0
ListPciDevices=1

[Options]
NumberOfBuffers=10
SupportRoi=0
FeatureVisibility=0
DontStopInNonContinuous=1
DelayAfterStartAcq=50

[Device]
DeviceSettings=
DefaultExposure=10000
DefaultImageWidth=640
DefaultImageHeight=480
DefaultTriggerSource=Line1

[PCI-Interface-CheckSight-P32B]
Settings=C:\Prg\Simplon\Bin\PCI-Interface-CheckSight-P32B.cam
```

```
DefaultExposure=16019
DefaultImageWidth=656
DefaultImageHeight=492
```

8.3.2.1. Interfaces

In this section you can specify, which interfaces are to be scanned for devices. Typically, the Cognex software installs support for GigE camera, so if you enable `ListGigEDevices`, then the GigE camera would appear in the list twice – once through the GigE-Vision protocol, once through the Simplon (GenTL) interface. As this adapter is primarily designed for CheckSight, which is a PCI device, the `ListPciDevices=1` assures its device(s) will be listed.

8.3.2.2. Options

NumberOfBuffers: Defines how many buffers should be used by Simplon for the acquisition. The higher number of buffers may be needed in case your Cognex application may not pick in time the acquired images; they are queued and the higher number of images means a longer queue. The default value is 10, in case you use triggered or non-continuous acquisition and pick images in time, the number of buffers can be reduced to 3.

SupportRoi: Enables the support of hardware ROI. This should be enabled only if the given sensor supports the hardware ROI (AOI). You can check this in Simplon Explorer – in such case the `LvDevice_LvAOIMode` feature can be set to `LvAOIMode_Manual`, and `LvDevice_OffsetX`, `LvDevice_OffsetY` can be set to a non-zero value. On CheckSight the support of hardware ROI is useful only in case the sensor provides faster acquisition with reduced ROI; on GigE cameras the HW ROI can also save network traffic. If the ROI is not hardware supported, the Cognex libraries make the ROI automatically by software.

FeatureVisibility: The device features, as you can see them in Simplon Explorer, are also available as Features of the imaging device in the Cognex library. Enumeration of the features consumes some time, so if you want to reduce the device connection time, you can limit the available features by selecting the level. The levels correspond to the levels visible in Simplon Explorer. The lower level, the faster is the connection of the device. Beginner = 0, Expert = 1, Guru = 2. See also the Features chapter.

DontStopInNonContinuous: When in non-continuous mode, an application usually calls `StartAcquisition()` function and after obtaining the image it calls the `StopAcquisition()` function. Starting and stopping the acquisition is time consuming, so the adapter introduces a mode, in which it postpones the StopAcquisition() call to the moment when it is really needed. If the next `StartAcquisition()` comes just after `StopAcquisition()` without change of any feature or other parameters, these 2 calls are skipped (only the acquisition of a new image is triggered) and time is saved. This enhancement is available only in non-continuous and non-triggered mode. You can disable it by this option.

8.3.2.3. Device

DeviceSettings: Here you can specify the name of the settings file for the device. You can create this file in Simplon Explorer by saving the settings in menu "Camera -> Save camera settings". This enables you to load device settings, which you can comfortably set in Simplon Explorer. Common things like white balance, gain etc., can be set in Simplon Explorer and saved to this file. Note that the attempt to load these settings for a different device may cause an error. The file must be specified with a full path.

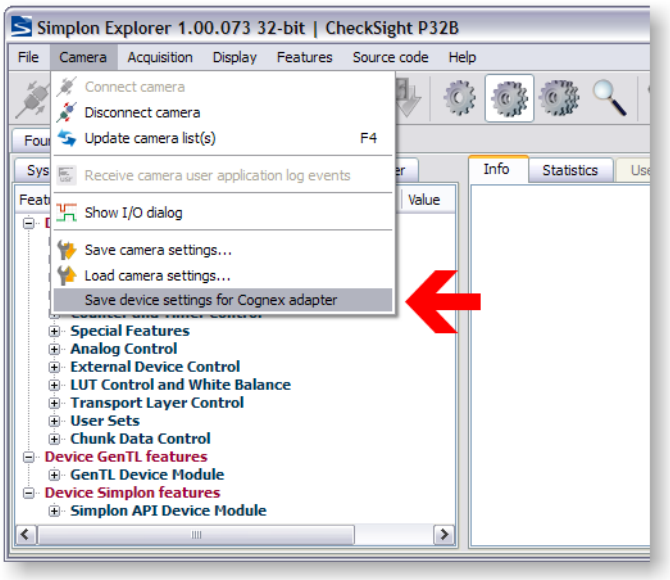
DefaultExposure, **DefaultImageWidth**, **DefaultImageHeight**: The Cognex software requires to obtain these values *before* the device is opened. Thus they are read from this file.

DefaultTriggerSource: As Leutron Vision devices usually have more trigger inputs, you can specify here which one will be used when the HW trigger is configured in Cognex software. On CheckSight the following inputs are available: Line1 - Opto In, Line2 - Opto In, Line3 - Opto In,

Line4 - Opto In, Line17 - TTL In, Line18 - TTL In. You can also set the trigger input by a feature, see the next chapter.

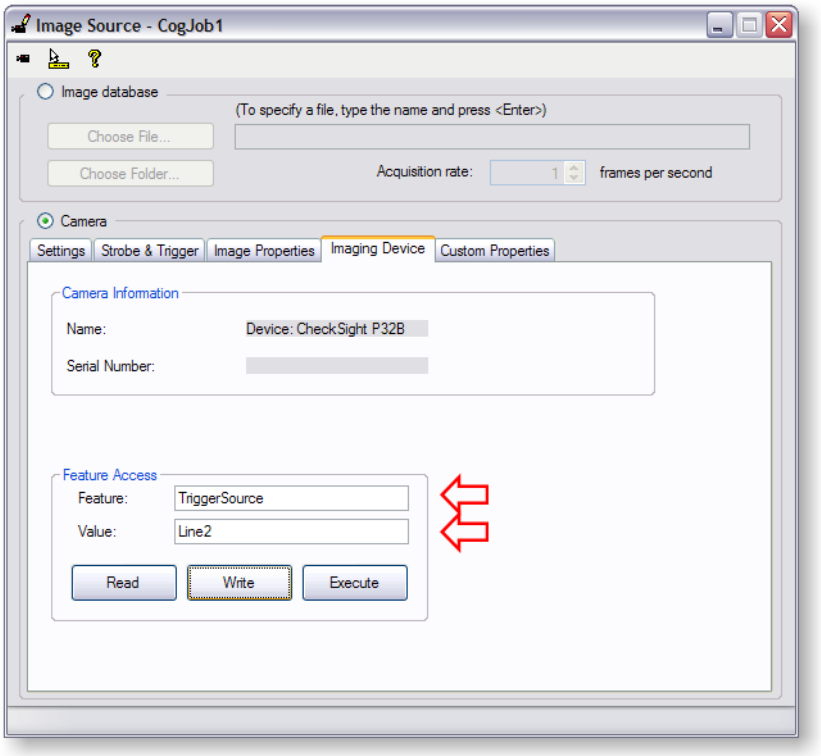
8.3.3. Settings saved from Simplon Explorer

The part printed in italics shows an example of settings, which can be saved easily from Simplon Explorer by the menu "Camera -> Save device settings for Cognex adapter". The section name is composed from Interface ID and Device ID, so the contents of this section is used only in case the same device is opened. If this section exists for the device, it has a precedence over the [Device] section contents.

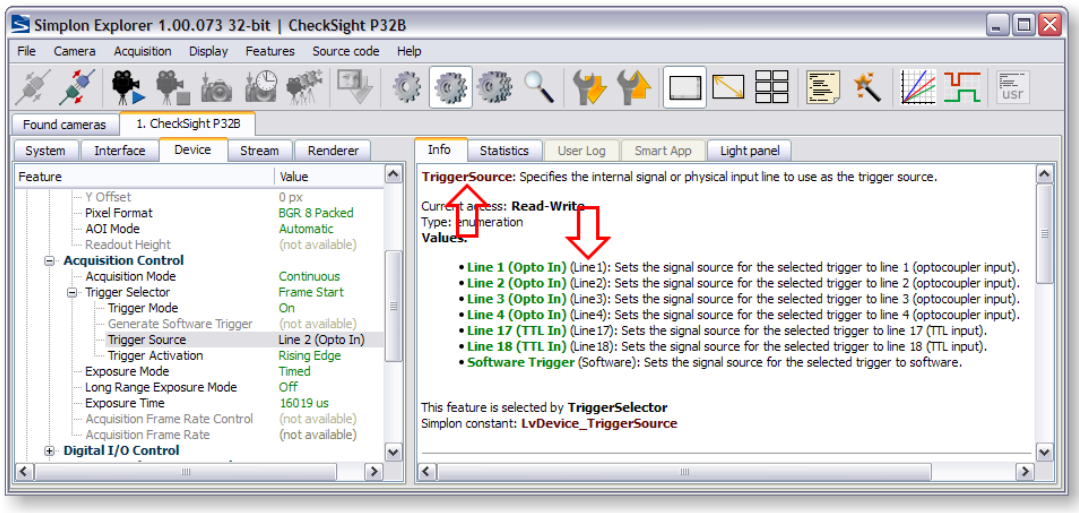


8.3.4. Features

The Device features visible in Simplon Explorer are also available through the Adapter. The feature is identified by its Gen-i-cam name and in case of enumerations you can also see the available values in Simplon Explorer. For example for setting the trigger source, the feature name is TriggerSource, and value is Line1, see the arrows in the screen shot. See also FeatureVisibility in the INI file.



For example in VisionPro QuickBuild you put these strings in the feature access dialog:


Simplon Explorer 1.00.073 32-bit | CheckSight P32B
File
Camera
Acquisition
Display
Features
Source code
Help
Found cameras
1. CheckSight P32B
System
Interface
Device
Stream
Renderer
Feature
Value
Y Offset
Pixel Format
AOI Mode
Readout Height
Acquisition Control
Acquisition Mode
Trigger Selector
Trigger Mode
Generate Software Trigger
Trigger Source
Trigger Activation
Exposure Mode
Long Range Exposure Mode
Exposure Time
Acquisition Frame Rate Control
Acquisition Frame Rate
Digital I/O Control
Info
Statistics
User Log
Smart App
Light panel
TriggerSource: Specifies the internal signal or physical input line to use as the trigger source.
Current access: Read-Write
Type: enumeration
Values:
Line 1 (Opto In) (Line1): Sets the signal source for the selected trigger to line 1 (optocoupler input).
Line 2 (Opto In) (Line2): Sets the signal source for the selected trigger to line 2 (optocoupler input).
Line 3 (Opto In) (Line3): Sets the signal source for the selected trigger to line 3 (optocoupler input).
Line 4 (Opto In) (Line4): Sets the signal source for the selected trigger to line 4 (optocoupler input).
Line 17 (TTL In) (Line17): Sets the signal source for the selected trigger to line 17 (TTL input).
Line 18 (TTL In) (Line18): Sets the signal source for the selected trigger to line 18 (TTL input).
Software Trigger (Software): Sets the signal source for the selected trigger to software.
This feature is selected by TriggerSelector
Simplon constant: LvDevice_TriggerSource

8.4. OpenCV

The OpenCV library does not provide native support for GenTL or GigE Vision interfaces, however, it is possible to integrate OpenCV with Simplon API easily. A simple example demonstrating, how to deliver the images acquired through the Simplon API interface to the OpenCV functions, can be created through the Simplon Source Code Generator — use the OpenCV sample in the Project Generation Wizard:

Copyright © Leutron Vision

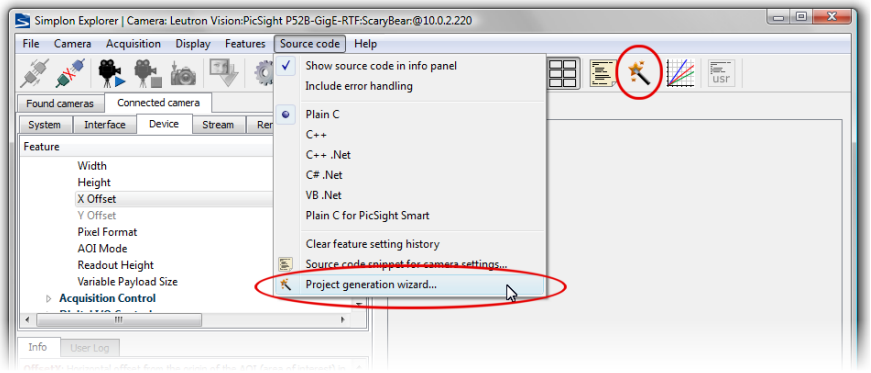


Figure 8.1. Simplon Explorer — start the Project Generation Wizard

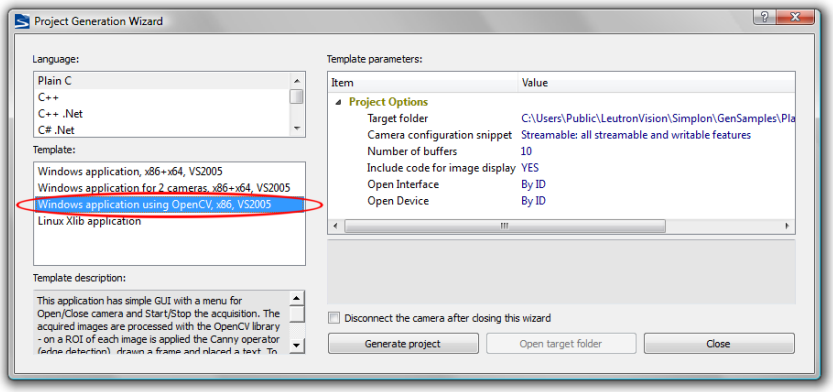


Figure 8.2. Simplon Explorer — select the OpenCV template in the wizard

The generated sample is trivial and should be easy to follow. It passes the acquired data to the OpenCV library using its `IplImage` structure. This approach allows to keep the full flexibility of the Simplon API interface while programming with OpenCV.

The OpenCV library can be used also on our PicSight-Smart architecture.

9. Troubleshooting & Support

9.1. Getting support for Leutron Vision products

9.1.1. Before you contact the support team

To get the most effective support for your problems, please make sure that you

- Check if your issue does not belong to the list of the most common issues (providing solutions to all of them).
- Send the problem report to proper e-mail address, depending on your location and the problem nature, as explained in Section 9.1.2, "Contact options" [p. 234].
- Make sure the problem is well reproducible, preferably with the Simplon Explorer and that you capture the `lv.simplon.log` file (Section 9.1.3, "Gathering information about the problem" [p. 234]).
- Attach the captured log file (one per each session if multiple test scenarios are described in the report) as well as other important information, describe all details about the problem, see Section 9.1.4, "Reporting the problem" [p. 235].

Following these guidelines will guarantee that your problem can be solved effectively, in the shortest possible time.

9.1.2. Contact options

Following list should help to identify the e-mail contact to be used for your problem report:

- If you purchased Leutron Vision products through a distributor, please contact that distributor with every support case. The list of Leutron Vision distributors[1] with their contact information is provided in our website.
- If you reside in Germany or Austria, contact <desupport@leutron.com>.
- If you reside in US or Canada, contact <ussupport@leutron.com>.
- If you reside in Czech republic or Slovakia, contact <czsupport@leutron.com>.
- In all other cases contact
  - <gigesupport@picsight.com> for issues related to PicSight-GigE/PicSight-Smart cameras
  - <linuxsupport@leutron.com> for any issues with Simplon for Linux
  - and <intsupport@leutron.com> for any other products, issues and questions

9.1.3. Gathering information about the problem

When collecting information about your problem, please follow the steps listed hereafter:

1. **Be sure you use the latest version of Simplon.** It may be that you have an old version of Simplon installed and you are experiencing a bug, which was already removed in the meantime. The version of currently installed Simplon is visible in the Simplon Explorer tool (menu Help → About). The latest Simplon version can be downloaded from the Simplon

[1] http://www.leutron.com/sales-contact/

download area[2]. If you have not registered yet for Simplon download on our website, you will have to do so first.

2. **Enable logging.** Before attempting to reproduce the problem, be sure the logging is enabled through the Simplon Settings tool (or directly the `lv.simplon.ini` file (refer to Section 5.3.1, "Simplon Settings utility and the configuration file" [p. 76]). Enable logging to file, preferably with highest log level. If the application crashes and the log is not complete, switch on the `Disable all buffering` log configuration option.

3. **Test the hardware functionality with a supplied demo program, Simplon Explorer.** As the first step use the "standard" demo program, Simplon Explorer, which covers most of the possibilities of Leutron Vision hardware usage.

   If you are able to reproduce the problem in Simplon Explorer, then the problem is probably not in your code. However, it still need not be a bug, a problem can be also caused by an improper configuration.

4. **Camera configuration.** Check especially whether the camera is set in a proper working mode. See camera documentation to learn how the camera should be adjusted.

5. **Check with another hardware.** Sometimes the problem can be caused by the HW/SW environment or by a defective piece of hardware. If it is in your possibilities, try if the same problem happens on another PC. If you have another camera of the same type available, try if the problem happens also after the camera replacement. For GigE Vision based cameras, review the network configuration and try to connect the camera directly, without additional network components in the path.

6. **Check with your application.** If the problem could not be reproduced in the previous steps, reproduce the problem in your application. If it is possible, extract and send us the smallest part of your code that can demonstrate where the problem occurs. Even better is if you can make the extracted code fragment compilable, so that the problem can be reproduced with a small application that you can send to us. Make sure that your application does proper error handling when accessing Simplon functions. Ignoring the error status of Simplon functions can lead to unexpected behavior of your application.

7. **Screenshots.** In case of disturbances in the acquired images, make and send us the screenshots of them, either as bitmaps with lossless compression (`.bmp`, `.png`) or as `.jpg` with a high quality (small compression).

8. **Send us the report.** Send all the collected information (and especially the `lv.simplon.log` file) to us, following guidelines from Section 9.1.4, "Reporting the problem" [p. 235].

9.1.4. Reporting the problem

Each problem report should contain following information:

- The e-mail subject should give a brief but specific description of the problem and your company name (separated by a "@" character). The subject will be used to identify evolution of the support case, so if you need to submit a different problem, create a new subject string, do not reuse the old one.
  - Example of a good subject: *Changing exposure time has no effect @ My Company Inc.*
  - Example of a bad subject: *Problem with camera*
- With each new support case, please list your Simplon registration number in the e-mail.

[2] http://www.leutron.com/support-downloads/download-area/

- Describe your problem in detail, including all its symptoms, conditions under which it occurs and all other information that can help resolving the problem. List all the information you gathered while testing and reproducing the problem, as described in Section 9.1.3, "Gathering information about the problem" [p. 234].
- Attach the required `lv.simplon.log` file (one per each reported session). The file absolutely essential for our support team and failing to attach it will usually only lead to a response asking you to collect and send it, which will in effect make the time needed for the resolution longer. Be sure that you send the log file generated during the problematic session you are mentioning in your report reporting.
- If troubleshooting a CheckSight camera running Linux, attach also the device driver log (Section 9.2.3.1, "Linux driver logging" [p. 237]) and contents of `/proc/lvsm`. The driver logs are appended to the standard kernel log and can be retrieved for example using the `dmesg` command.
- If the problem occurs only under some conditions or only with some particular piece of hardware or software, describe the differences between the two cases and attach log files generated during both the successful and problematic sessions. If only a specific sequence of steps leads to the problem, describe this sequence.

9.1.5. Return Material Authorization (RMA)

If you need to return some (possibly defective) material to Leutron Vision, please use the RMA application[3] on our website. The application will query you for some important information about your product(s) and their defect(s), deciding automatically, whether you should send the material immediately back to Leutron Vision (or your distributor) or whether we will first contact you to do more tests for problem verification.

9.1.6. Customer registration

Customers willing to get (different level of) access rights to Leutron Vision download area[4] should first register on our website. The registration area[5] provides various registration possibilities from access to freely downloadable data (anyone can get access to it) to access to Simplon download[6] (you need to purchase Simplon first).

9.2. Troubleshooting Leutron Vision products

9.2.1. Troubleshooting common issues

The lists below mention some most common issues that can be faced by customers, but have easy solution. It is a good idea to check this list first when encountering a problem. It is also a good idea to check Section 9.2.2, "Known problems and limitations" [p. 237] and Section 9.2.3, "Simplon logging" [p. 237] whether some hints can be found there.

- If you are not getting any image, if you get different image data than expected or if you have problems with acquisition timing, check again carefully that the camera is set in proper working mode and that it is not in some error status (see camera manual for possible description of such situations).
- If you have problems with acquisition from a PicSight-GigE/PicSight-Smart camera, check whether your firewall settings do not interfere with the acquisition. PicSight-GigE user manual provides more information about this topic.

[3] http://www.leutron.com/rma/client_id.php
[4] http://www.leutron.com/support-downloads/download-area/
[5] http://www.leutron.com/support-downloads/registrations/
[6] http://www.leutron.com/support-downloads/download-area/

9.2.2. Known problems and limitations

9.2.2.1. Simplon software package

(none in current version)

9.2.2.2. CheckSight cameras

(none in current version)

9.2.3. Simplon logging

All the Simplon components are using a common logging subsystem, which can be configured through the `lv.simplon.ini` and Simplon Settings (Section 5.3.1, “Simplon Settings utility and the configuration file” [p. 76]).

The log can be directed to various outputs, while the most important of them (especially for support) is the file output. The log file path can be adjusted, however the default name (`lv.simplon.log`) as well as the default location (application data directory under Windows, `/var/opt/simplon/log` under Linux) will be just right in most situations.

The logging level can also be adjusted. While the default level `Info (4)` provides most useful information about the system and Simplon operation, for specific troubleshooting and support requests, it is often useful to switch to the most verbose level `Trace (6)`.

For the case of troubleshooting problems involving an application crash, when the log might not be written completely (log buffers not flushed completely at the time of application crash), the logging subsystem should be configured to avoid any buffering. In such case, please switch on the `Disable all buffering` option.

The logging configuration offers few more options, the most important of them being the max allowed size of the log file. The log can be also directed to other outputs, but those are mainly intended for use during advanced debugging by qualified personnel.

Several logging mechanisms are used by various Simplon components. They are separated to utilize log analysis, to separate domain-specific info from the general purpose logs and to improve performance, logging only what needs to get logged in a given situation. Creation of these log files can usually be controlled by the user.

Note however, that all the information in the logs is just informative and its main purpose is to assist the Leutron Vision support team when trying to resolve possible problems. It is not designed as a tool that should be necessarily well understandable by users, it might be misleading and all the information is context-specific. If a word “error” or “warning” appears in the log, it does not necessarily mean a real error occurred, it might be valid situation in a given context. You don't need to get alarmed because of that and contact support, especially if the application is otherwise working well.

9.2.3.1. Linux driver logging

On the CheckSight systems, a device driver controlling the acquisition device is running. The driver can also provide useful logs, but those are not routed through Simplon user level logging subsystem, but rather through standard Linux kernel logging mechanisms. Depending on the system configuration, there might be multiple ways to access that log, but the most convenient is usually to simply execture the `dmesg` command and store its output to a file.

Apart of the kernel log messages, other (mostly static) information can be available through reading the `/proc/lvsm`.

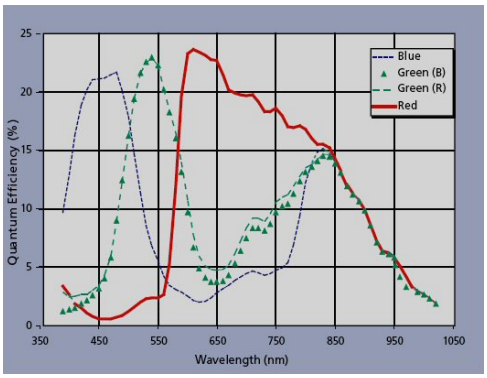
9.2.4. Debugging symbols

When debugging certain kinds of problems, such as application crash, it might be very useful if you can provide the support team with extended diagnostic information, such as the stack trace showing the crash point. The installer of Simplon for Windows allows to install “PDB files” — files containing the debugging information (symbols) for the Simplon libraries. We recommend to install these debug files on the development and testing systems, so that it is possible to generate the stack trace info whenever encountering a problem. The files are located in the `bin\Debug` subdirectory of the Simplon installation.

Note that in Linux version of Simplon the symbols are directly included in the libraries and executables.

Appendix A. Technical specifications for individual models

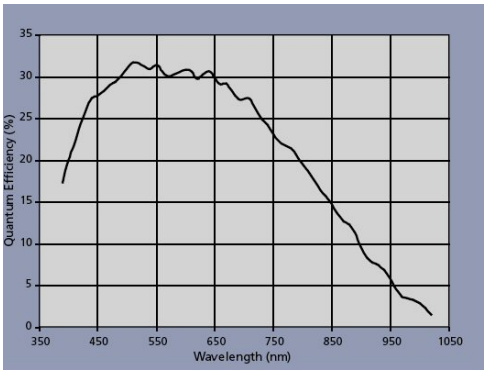
A.1. CheckSight G32B

Specification	CheckSight G32B
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/2" CMOS, parallel readout, type Aptina MT9V403
Spectral sensitivity	
Monochrome/color	bayer pattern
Output image resolution (H×V)	656 × 492 pixels
Frame rate	196 fps
Video format	10-bit, single tap
Scanning	progressive
Pixel size (H×V)	9.9 × 9.9 µm
Sensor size (H×V)	6.5241 × 4.8906 mm
Shutter	global, 20 µs –1300 ms (default 5 ms)
Operating (trigger) modes	• Free running • Triggered mode
Gain & black level	• Analog gain: 0 .. 23.52 dB (default 0 dB) • Digital gain: n/a • Black level: -1.5 .. 1.5 DN (default 0 DN)
Auto functions	n/a
Signal/noise ratio	46 dB
Dynamic range	53 dB
Area of interest (AOI)	Horizontal and vertical AOI applicable directly at the sensor
Binning	n/a
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g

Specification	CheckSight G32B
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 401801: CheckSight G32B/4GB • 401802: CheckSight G32B/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.1. CheckSight G32B technical specifications

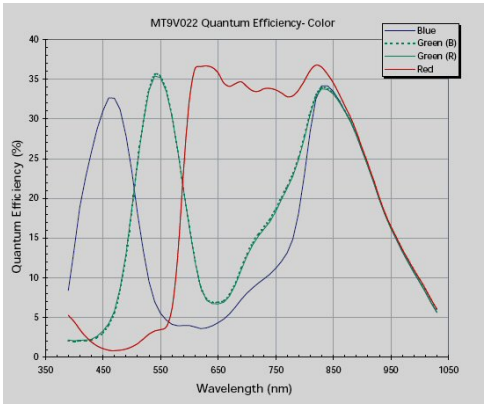
A.2. CheckSight G32M

Specification	CheckSight G32M
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/2" CMOS, parallel readout, type Aptina MT9V403
Spectral sensitivity	
Monochrome/color	monochrome
Output image resolution (H×V)	656 × 492 pixels
Frame rate	196 fps
Video format	10-bit, single tap
Scanning	progressive
Pixel size (H×V)	9.9 × 9.9 µm
Sensor size (H×V)	6.5241 × 4.8906 mm
Shutter	global, 20 µs –1300 ms (default 5 ms)
Operating (trigger) modes	• Free running • Triggered mode
Gain & black level	• Analog gain: 0 .. 23.52 dB (default 0 dB) • Digital gain: n/a • Black level: -1.5 .. 1.5 DN (default 0 DN)

Specification	CheckSight G32M
Auto functions	n/a
Signal/noise ratio	46 dB
Dynamic range	53 dB
Area of interest (AOI)	Horizontal and vertical AOI applicable directly at the sensor
Binning	n/a
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 400801: CheckSight G32M/4GB • 400802: CheckSight G32M/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.2. CheckSight G32M technical specifications

A.3. CheckSight G43AB

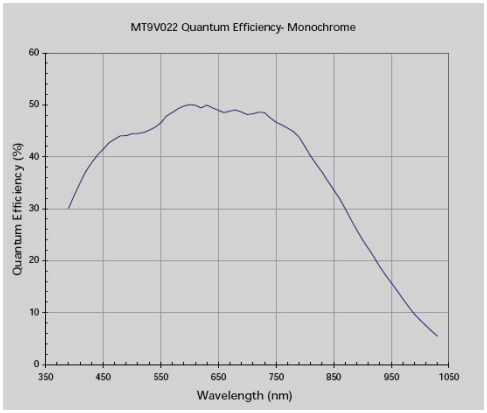
Specification	CheckSight G43AB
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/3" CMOS, parallel readout, type Aptina MT9V033
Spectral sensitivity	
Monochrome/color	bayer pattern
Output image resolution (H×V)	752 × 480 pixels
Frame rate	62 fps
Video format	10-bit, single tap

Specification	CheckSight G43AB
Scanning	progressive
Pixel size (H×V)	6 × 6 µm
Sensor size (H×V)	4.512 × 2.88 mm
Shutter	global, 10 µs –1000 ms (default 15 ms)
Operating (trigger) modes	• Free running • Triggered mode
Gain & black level	• Analog gain: 0 .. 12.04 dB (default 12 dB) • Digital gain: -12.04 .. 11.48 dB (default 0 dB) • Black level: 0 .. 6.35 DN (default 0 DN)
Auto functions	• Auto exposure • Automatic analog gain control • Automatic black level
Signal/noise ratio	42 dB
Dynamic range	49 dB
Area of interest (AOI)	Horizontal and vertical AOI applicable directly at the sensor
Binning	n/a
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 413801: CheckSight G43AB/4GB • 413802: CheckSight G43AB/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.3. CheckSight G43AB technical specifications

A.4. CheckSight G43AM

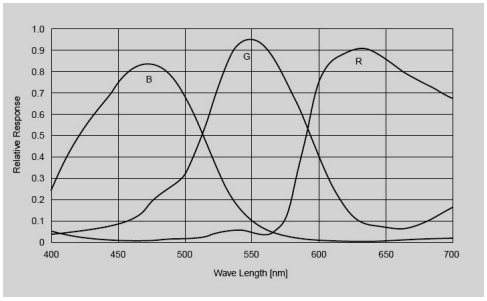
Specification	CheckSight G43AM
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/3" CMOS, parallel readout, type Aptina MT9V033

Specification	CheckSight G43AM
Spectral sensitivity	
Monochrome/color	monochrome
Output image resolution (H×V)	752 × 480 pixels
Frame rate	62 fps
Video format	10-bit, single tap
Scanning	progressive
Pixel size (H×V)	6 × 6 µm
Sensor size (H×V)	4.512 × 2.88 mm
Shutter	global, 10 µs –1000 ms (default 15 ms)
Operating (trigger) modes	• Free running • Triggered mode
Gain & black level	• Analog gain: 0 .. 12.04 dB (default 4 dB) • Digital gain: -12.04 .. 11.48 dB (default 0 dB) • Black level: 0 .. 6.35 DN (default 0 DN)
Auto functions	• Auto exposure • Automatic analog gain control • Automatic black level
Signal/noise ratio	42 dB
Dynamic range	49 dB
Area of interest (AOI)	Horizontal and vertical AOI applicable directly at the sensor
Binning	• Horizontal: 2× 4× • Vertical: 2× 4×
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV

Specification	CheckSight G43AM
Ordering information	• 412801: CheckSight G43AM/4GB • 412802: CheckSight G43AM/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.4. CheckSight G43AM technical specifications

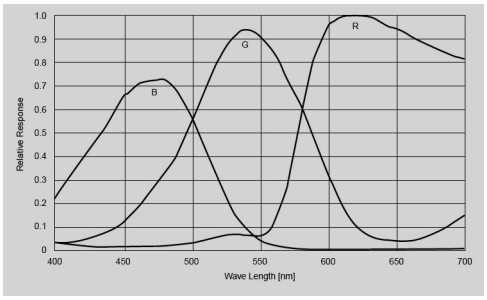
A.5. CheckSight P133B

Specification	CheckSight P133B
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/1" CCD, interline transfer, type SONY ICX442AQ
Spectral sensitivity	
Monochrome/color	bayer pattern
Output image resolution (H×V)	1304 × 976 pixels
Full sensor resolution (incl. black pixels)	1360 × 986 pixels
Frame rate	8.3 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	0 × 0 µm
Sensor size (H×V)	6.23 × 5.09 mm
Shutter	global, 5 µs –60000 ms (default 100 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode
Gain & black level	• Analog gain: -10 .. 26 dB (default 0 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	n/a
Decimation	n/a

Specification	CheckSight P133B
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 151801: CheckSight P133B/4GB • 151802: CheckSight P133B/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.5. CheckSight P133B technical specifications

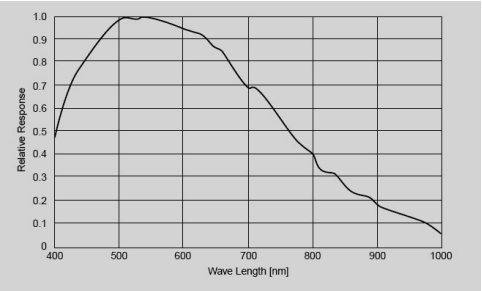
A.6. CheckSight P141B

Specification	CheckSight P141B
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	2/3" CCD, interline transfer, type SONY ICX285
Spectral sensitivity	
Monochrome/color	bayer pattern
Output image resolution (H×V)	1392 × 1040 pixels
Full sensor resolution (incl. black pixels)	1438 × 1050 pixels
Frame rate	20.2 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	6.45 × 6.45 µm
Sensor size (H×V)	10.2 × 8.3 mm
Shutter	global, 5 µs –60000 ms (default 48 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode

Specification	CheckSight P141B
Gain & black level	• Analog gain: -3 .. 33 dB (default 9 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	n/a
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 181801: CheckSight P141B/4GB • 181802: CheckSight P141B/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.6. CheckSight P141B technical specifications

A.7. CheckSight P141M

Specification	CheckSight P141M
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	2/3" CCD, interline transfer, type SONY ICX285
Spectral sensitivity	
Monochrome/color	monochrome
Output image resolution (H×V)	1392 × 1040 pixels
Full sensor resolution (incl. black pixels)	1438 × 1050 pixels
Frame rate	20.2 fps
Video format	12-bit, single tap

Specification	CheckSight P141M
Scanning	progressive
Pixel size (H×V)	6.45 × 6.45 µm
Sensor size (H×V)	10.2 × 8.3 mm
Shutter	global, 5 µs –60000 ms (default 48 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode
Gain & black level	• Analog gain: -3 .. 33 dB (default 0 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Signal/noise ratio	43 dB
Dynamic range	77 dB
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	• Vertical: 2×
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 180801: CheckSight P141M/4GB • 180802: CheckSight P141M/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.7. CheckSight P141M technical specifications

A.8. CheckSight P142AB

Specification	CheckSight P142AB
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/2" CCD, interline transfer, type SONY ICX267

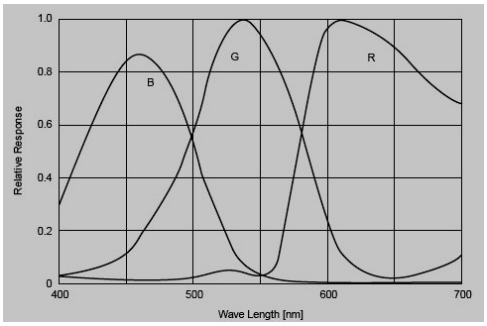
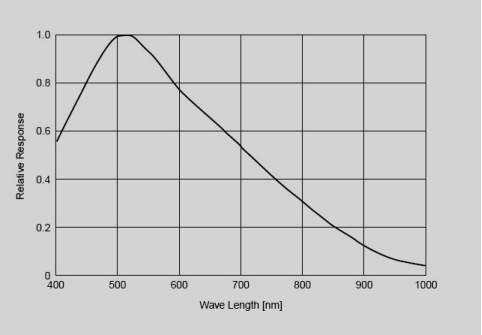
Specification	CheckSight P142AB
Spectral sensitivity	
Monochrome/color	bayer pattern
Output image resolution (H×V)	1392 × 1040 pixels
Full sensor resolution (incl. black pixels)	1434 × 1050 pixels
Frame rate	0.15 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	4.65 × 4.65 µm
Sensor size (H×V)	7.6 × 6.2 mm
Shutter	global, 5 µs –60000 ms (default 130 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode
Gain & black level	• Analog gain: -8 .. 28 dB (default 0 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	n/a
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 171801: CheckSight P142AB/4GB • 171802: CheckSight P142AB/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.8. CheckSight P142AB technical specifications

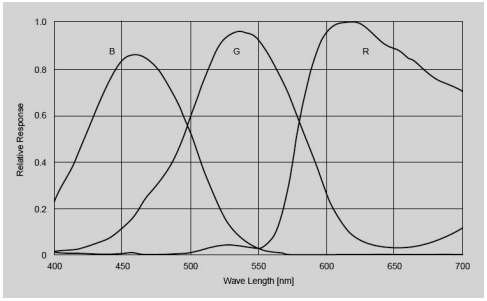
A.9. CheckSight P142AM

Specification	CheckSight P142AM
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/2" CCD, interline transfer, type SONY ICX267
Spectral sensitivity	
Monochrome/color	monochrome
Output image resolution (H×V)	1392 × 1040 pixels
Full sensor resolution (incl. black pixels)	1434 × 1050 pixels
Frame rate	0.15 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	4.65 × 4.65 µm
Sensor size (H×V)	7.6 × 6.2 mm
Shutter	global, 5 µs –60000 ms (default 50 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode
Gain & black level	• Analog gain: -8 .. 28 dB (default 0 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Signal/noise ratio	43 dB
Dynamic range	71 dB
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	• Vertical: 2×
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface

Specification	CheckSight P142AM
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 170801: CheckSight P142AM/4GB • 170802: CheckSight P142AM/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.9. CheckSight P142AM technical specifications

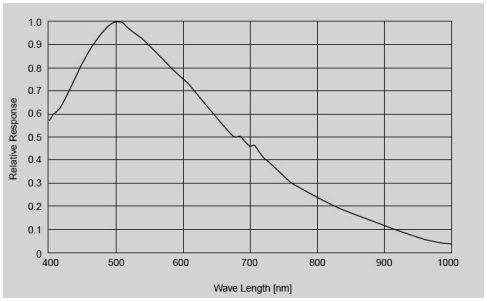
A.10. CheckSight P202B

Specification	CheckSight P202B
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/1.8" CCD, interline transfer, type SONY ICX274
Spectral sensitivity	
Monochrome/color	bayer pattern
Output image resolution (H×V)	1624 × 1236 pixels
Full sensor resolution (incl. black pixels)	1688 × 1248 pixels
Frame rate	14.6 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	4.4 × 4.4 µm
Sensor size (H×V)	8.5 × 6.8 mm
Shutter	global, 5 µs –60000 ms (default 60 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode
Gain & black level	• Analog gain: -10 .. 26 dB (default 4 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Area of interest (AOI)	Vertical AOI applicable directly at the sensor

Specification	CheckSight P202B
Binning	n/a
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 211801: CheckSight P202B/4GB • 211802: CheckSight P202B/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.10. CheckSight P202B technical specifications

A.11. CheckSight P202M

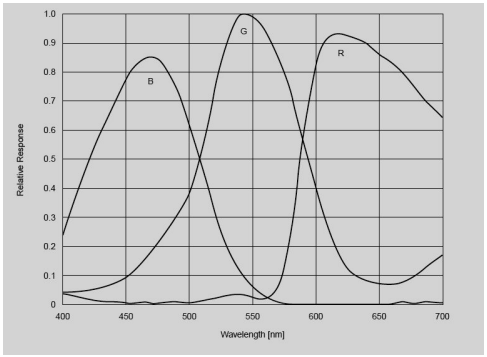
Specification	CheckSight P202M
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/1.8" CCD, interline transfer, type SONY ICX274
Spectral sensitivity	
Monochrome/color	monochrome
Output image resolution (H×V)	1624 × 1236 pixels
Full sensor resolution (incl. black pixels)	1688 × 1248 pixels
Frame rate	14.6 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	4.4 × 4.4 µm
Sensor size (H×V)	8.5 × 6.8 mm
Shutter	global, 5 µs –60000 ms (default 30 ms)

Specification	CheckSight P202M
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode
Gain & black level	• Analog gain: -10 .. 26 dB (default 0 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Signal/noise ratio	40 dB
Dynamic range	69 dB
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	• Vertical: 2×
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 210801: CheckSight P202M/4GB • 210802: CheckSight P202M/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.11. CheckSight P202M technical specifications

A.12. CheckSight P203B

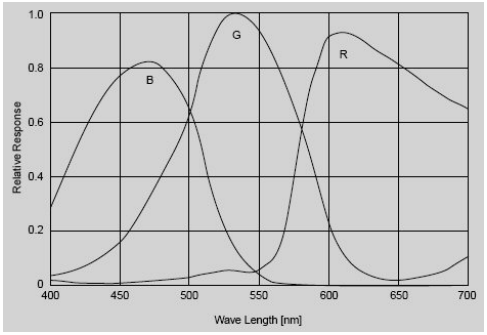
Specification	CheckSight P203B
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/1" CCD, interline transfer, type SONY ICX454JQF-J

Specification	CheckSight P203B
Spectral sensitivity	
Monochrome/color	bayer pattern
Output image resolution (H×V)	1648 × 1240 pixels
Full sensor resolution (incl. black pixels)	1690 × 1250 pixels
Frame rate	9.2 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	0 × 0 µm
Sensor size (H×V)	6.38 × 5.26 mm
Shutter	global, 5 µs –60000 ms (default 100 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode
Gain & black level	• Analog gain: -6 .. 26 dB (default 6 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	n/a
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV

Specification	CheckSight P203B
Ordering information	• 201801: CheckSight P203B/4GB • 201802: CheckSight P203B/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.12. CheckSight P203B technical specifications

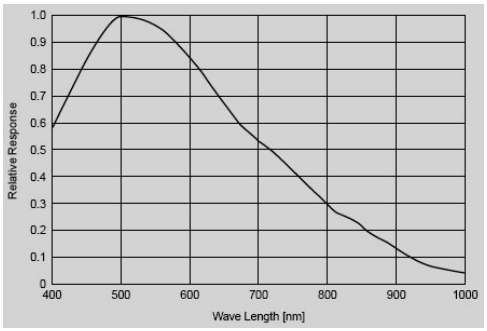
A.13. CheckSight P32B

Specification	CheckSight P32B
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/2" CCD, interline transfer, type SONY ICX414
Spectral sensitivity	
Monochrome/color	bayer pattern
Output image resolution (H×V)	656 × 492 pixels
Full sensor resolution (incl. black pixels)	692 × 504 pixels
Frame rate	62.3 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	9.9 × 9.9 µm
Sensor size (H×V)	7.48 × 6.15 mm
Shutter	global, 5 µs –60000 ms (default 16 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode
Gain & black level	• Analog gain: -6 .. 30 dB (default 10 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	n/a
Decimation	n/a

Specification	CheckSight P32B
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 121801: CheckSight P32B/4GB • 121802: CheckSight P32B/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.13. CheckSight P32B technical specifications

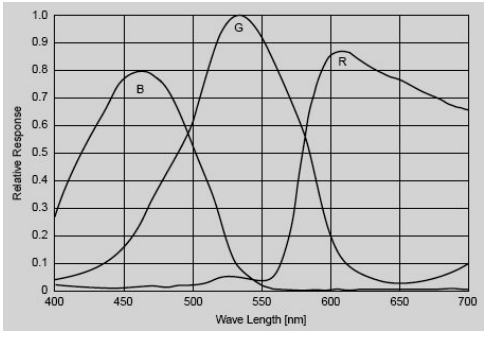
A.14. CheckSight P32M

Specification	CheckSight P32M
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/2" CCD, interline transfer, type SONY ICX414
Spectral sensitivity	
Monochrome/color	monochrome
Output image resolution (H×V)	656 × 492 pixels
Full sensor resolution (incl. black pixels)	692 × 504 pixels
Frame rate	62.3 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	9.9 × 9.9 µm
Sensor size (H×V)	7.48 × 6.15 mm
Shutter	global, 5 µs –60000 ms (default 16 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode

Specification	CheckSight P32M
Gain & black level	• Analog gain: -6 .. 30 dB (default 4 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Signal/noise ratio	45 dB
Dynamic range	73 dB
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	• Vertical: 2×
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 120801: CheckSight P32M/4GB • 120802: CheckSight P32M/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.14. CheckSight P32M technical specifications

A.15. CheckSight P33B

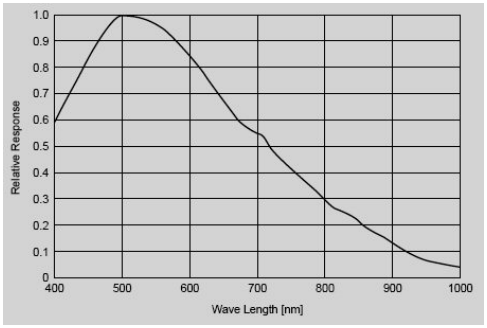
Specification	CheckSight P33B
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/3" CCD, interline transfer, type SONY ICX424
Spectral sensitivity	
Monochrome/color	bayer pattern
Output image resolution (H×V)	656 × 492 pixels

Specification	CheckSight P33B
Full sensor resolution (incl. black pixels)	692 × 504 pixels
Frame rate	62.3 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	7.4 × 7.4 µm
Sensor size (H×V)	5.79 × 4.89 mm
Shutter	global, 5 µs –60000 ms (default 16 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode
Gain & black level	• Analog gain: -6 .. 30 dB (default 12 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	n/a
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 111801: CheckSight P33B/4GB • 111802: CheckSight P33B/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.15. CheckSight P33B technical specifications

A.16. CheckSight P33M

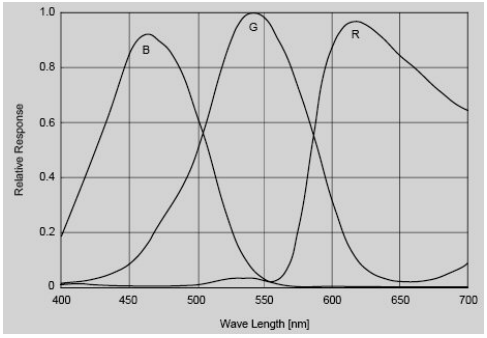
Specification	CheckSight P33M
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/3" CCD, interline transfer, type SONY ICX424

Specification	CheckSight P33M
Spectral sensitivity	
Monochrome/color	monochrome
Output image resolution (H×V)	656 × 492 pixels
Full sensor resolution (incl. black pixels)	692 × 504 pixels
Frame rate	62.3 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	7.4 × 7.4 µm
Sensor size (H×V)	5.79 × 4.89 mm
Shutter	global, 5 µs –60000 ms (default 16 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode
Gain & black level	• Analog gain: -6 .. 30 dB (default 4 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Signal/noise ratio	40 dB
Dynamic range	63 dB
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	• Vertical: 2×
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV

Specification	CheckSight P33M
Ordering information	• 110801: CheckSight P33M/4GB • 110802: CheckSight P33M/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.16. CheckSight P33M technical specifications

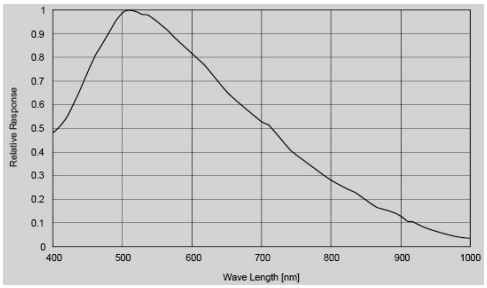
A.17. CheckSight P34B

Specification	CheckSight P34B
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/4" CCD, interline transfer, type SONY ICX098
Spectral sensitivity	
Monochrome/color	bayer pattern
Output image resolution (H×V)	656 × 492 pixels
Full sensor resolution (incl. black pixels)	692 × 504 pixels
Frame rate	64.7 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	5.6 × 5.6 µm
Sensor size (H×V)	4.6 × 3.97 mm
Shutter	global, 5 µs –60000 ms (default 15 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode
Gain & black level	• Analog gain: -6 .. 30 dB (default 16 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	n/a
Decimation	n/a

Specification	CheckSight P34B
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 101801: CheckSight P34B/4GB • 101802: CheckSight P34B/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.17. CheckSight P34B technical specifications

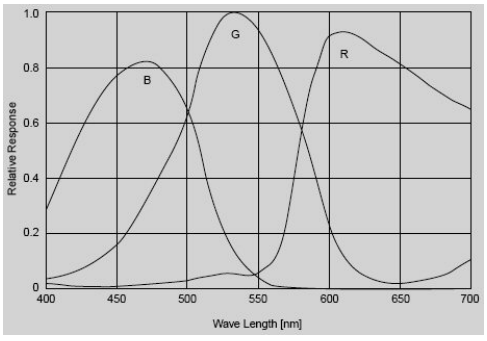
A.18. CheckSight P34M

Specification	CheckSight P34M
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/4" CCD, interline transfer, type SONY ICX098
Spectral sensitivity	
Monochrome/color	monochrome
Output image resolution (H×V)	656 × 492 pixels
Full sensor resolution (incl. black pixels)	692 × 504 pixels
Frame rate	64.7 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	5.6 × 5.6 µm
Sensor size (H×V)	4.6 × 3.97 mm
Shutter	global, 5 µs –60000 ms (default 15 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode

Specification	CheckSight P34M
Gain & black level	• Analog gain: -6 .. 30 dB (default 9 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Signal/noise ratio	44 dB
Dynamic range	69 dB
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	• Vertical: 2×
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 100801: CheckSight P34M/4GB • 100802: CheckSight P34M/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.18. CheckSight P34M technical specifications

A.19. CheckSight P52B

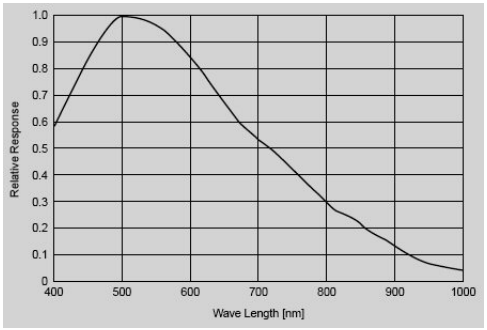
Specification	CheckSight P52B
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/2" CCD, interline transfer, type SONY ICX415
Spectral sensitivity	
Monochrome/color	bayer pattern
Output image resolution (H×V)	776 × 580 pixels

Specification	CheckSight P52B
Full sensor resolution (incl. black pixels)	823 × 592 pixels
Frame rate	62.3 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	8.3 × 8.3 µm
Sensor size (H×V)	7.48 × 6.15 mm
Shutter	global, 5 µs –60000 ms (default 15 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode
Gain & black level	• Analog gain: -10 .. 26 dB (default 12 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	n/a
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 131801: CheckSight P52B/4GB • 131802: CheckSight P52B/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.19. CheckSight P52B technical specifications

A.20. CheckSight P52M

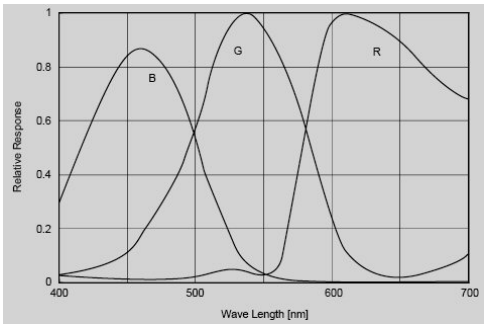
Specification	CheckSight P52M
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/2" CCD, interline transfer, type SONY ICX415

Specification	CheckSight P52M
Spectral sensitivity	
Monochrome/color	monochrome
Output image resolution (H×V)	776 × 580 pixels
Full sensor resolution (incl. black pixels)	823 × 592 pixels
Frame rate	62.3 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	8.3 × 8.3 µm
Sensor size (H×V)	7.48 × 6.15 mm
Shutter	global, 5 µs –60000 ms (default 15 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode
Gain & black level	• Analog gain: -10 .. 26 dB (default 4 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Signal/noise ratio	44 dB
Dynamic range	69 dB
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	• Vertical: 2×
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV

Specification	CheckSight P52M
Ordering information	• 130801: CheckSight P52M/4GB • 130802: CheckSight P52M/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.20. CheckSight P52M technical specifications

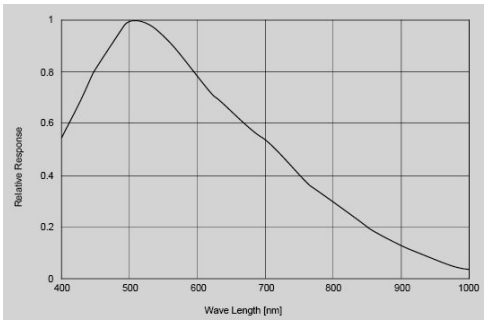
A.21. CheckSight P83B

Specification	CheckSight P83B
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/3" CCD, interline transfer, type SONY ICX204
Spectral sensitivity	
Monochrome/color	bayer pattern
Output image resolution (H×V)	1032 × 776 pixels
Full sensor resolution (incl. black pixels)	1077 × 788 pixels
Frame rate	32.5 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	4.65 × 4.65 µm
Sensor size (H×V)	5.8 × 4.92 mm
Shutter	global, 5 µs –60000 ms (default 30 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode
Gain & black level	• Analog gain: -8 .. 28 dB (default 16 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	n/a
Decimation	n/a

Specification	CheckSight P83B
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 141801: CheckSight P83B/4GB • 141802: CheckSight P83B/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.21. CheckSight P83B technical specifications

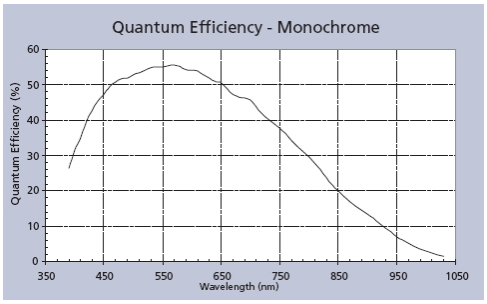
A.22. CheckSight P83M

Specification	CheckSight P83M
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/3" CCD, interline transfer, type SONY ICX204
Spectral sensitivity	
Monochrome/color	monochrome
Output image resolution (H×V)	1032 × 776 pixels
Full sensor resolution (incl. black pixels)	1077 × 788 pixels
Frame rate	32.5 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	4.65 × 4.65 µm
Sensor size (H×V)	5.8 × 4.92 mm
Shutter	global, 5 µs –60000 ms (default 30 ms)
Operating (trigger) modes	• Free running • Triggered mode • Long range exposure mode

Specification	CheckSight P83M
Gain & black level	• Analog gain: -8 .. 28 dB (default 14 dB) • Digital gain: n/a • Black level: 0 .. 25.5 DN (default 0 DN)
Auto functions	n/a
Signal/noise ratio	42 dB
Dynamic range	73 dB
Area of interest (AOI)	Vertical AOI applicable directly at the sensor
Binning	• Vertical: 2×
Decimation	n/a
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 140801: CheckSight P83M/4GB • 140802: CheckSight P83M/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.22. CheckSight P83M technical specifications

A.23. CheckSight R132M

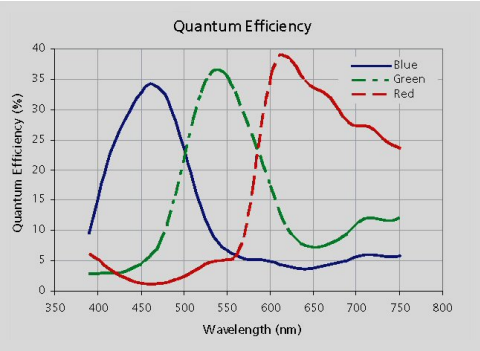
Specification	CheckSight R132M
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/2" CMOS, sequential readout, type Aptina MT9M001
Spectral sensitivity	
Monochrome/color	monochrom
Output image resolution (H×V)	1280 × 1024 pixels
Frame rate	30 fps

Specification	CheckSight R132M
Video format	10-bit, single tap
Scanning	progressive
Pixel size (H×V)	5.2 × 5.2 µm
Sensor size (H×V)	6.66 × 5.32 mm
Shutter	rolling, 28 µs –520 ms (default 30 ms)
Operating (trigger) modes	• Free running • Triggered mode
Gain & black level	• Analog gain: 0 .. 23.53 dB (default 14 dB) • Digital gain: n/a • Black level: -51.2 .. 51 DN (default 0 DN)
Auto functions	• Automatic black level
Signal/noise ratio	42 dB
Dynamic range	52 dB
Area of interest (AOI)	Horizontal and vertical AOI applicable directly at the sensor
Binning	n/a
Decimation	• Horizontal: 2× 4× 8× • Vertical: 2× 4× 8×
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 420801: CheckSight R132M/4GB • 420802: CheckSight R132M/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.23. CheckSight R132M technical specifications

A.24. CheckSight R312AB

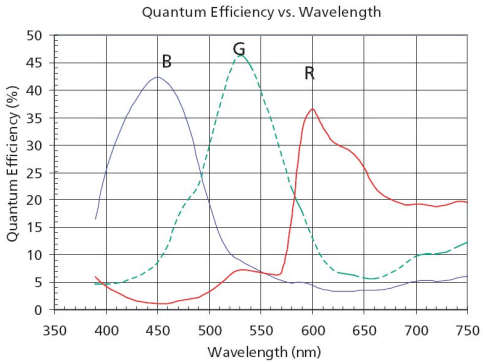
Specification	CheckSight R312AB
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/2" CMOS, sequential readout, type Aptina MT9T001

Specification	**CheckSight R312AB**
Spectral sensitivity	
Monochrome/color	bayer pattern
Output image resolution (H×V)	2048 × 1536 pixels
Frame rate	12.77 fps
Video format	10-bit, single tap
Scanning	progressive
Pixel size (H×V)	3.2 × 3.2 µm
Sensor size (H×V)	6.5536 × 4.9152 mm
Shutter	rolling, 47 µs –520 ms (default 10 ms)
Operating (trigger) modes	• Free running • Triggered mode
Gain & black level	• Analog gain: 0 .. 18.07 dB (default 18 dB) • Digital gain: 0 .. 24.08 dB (default 0 dB) • Black level: -51.2 .. 51 DN (default 0 DN)
Auto functions	• Automatic black level
Signal/noise ratio	43 dB
Dynamic range	61 dB
Area of interest (AOI)	Horizontal and vertical AOI applicable directly at the sensor
Binning	• Horizontal: 2× 3× • Vertical: 2× 3×
Decimation	• Horizontal: 2× 3× 4× 8× • Vertical: 2× 3× 4× 8×
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV

Specification	CheckSight R312AB
Ordering information	• 441801: CheckSight R312AB/4GB • 441802: CheckSight R312AB/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.24. CheckSight R312AB technical specifications

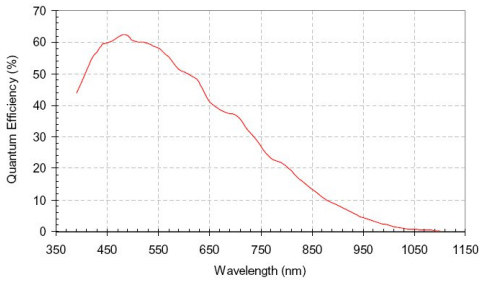
A.25. CheckSight R502B

Specification	CheckSight R502B
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/2.5" CMOS, sequential readout, type Aptina MT9P031
Spectral sensitivity	
Monochrome/color	bayer pattern
Output image resolution (H×V)	2592 × 1944 pixels
Frame rate	11.7 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	2.2 × 2.2 µm
Sensor size (H×V)	5.702 × 4.277 mm
Shutter	rolling, 39 µs –55000 ms (default 10 ms)
Operating (trigger) modes	• Free running • Triggered mode • Global reset mode
Gain & black level	• Analog gain: 0 .. 18.06 dB (default 0 dB) • Digital gain: 0 .. 24.08 dB (default 0 dB) • Black level: 0 .. 2.55 DN (default 0 DN)
Auto functions	• Automatic black level
Signal/noise ratio	38 dB
Dynamic range	70 dB
Area of interest (AOI)	Horizontal and vertical AOI applicable directly at the sensor

Specification	CheckSight R502B
Binning	• Horizontal: 2× 4× • Vertical: 2× 4×
Decimation	• Horizontal: 2× 3× 4× 5× 6× 7× • Vertical: 2× 3× 4× 5× 6× 7× 8×
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 451801: CheckSight R502B/4GB • 451802: CheckSight R502B/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.25. CheckSight R502B technical specifications

A.26. CheckSight R502M

Specification	CheckSight R502M
PC technology	Embedded PC with Intel Atom Z530, 1.6GHz, 1GB DDR2/533MHz, interface for CompactFlash card, Gigabit-Ethernet interface, 3×USB2 interfaces, 1 audio in- and 1 output (mono), DVI-D signal-compatible graphics output
Sensor	1/2.5" CMOS, sequential readout, type Aptina MT9P031
Spectral sensitivity	
Monochrome/color	monochrome
Output image resolution (H×V)	2592 × 1944 pixels
Frame rate	11.7 fps
Video format	12-bit, single tap
Scanning	progressive
Pixel size (H×V)	2.2 × 2.2 µm
Sensor size (H×V)	5.702 × 4.277 mm
Shutter	rolling, 39 µs –55000 ms (default 10 ms)

Specification	CheckSight R502M
Operating (trigger) modes	• Free running • Triggered mode • Global reset mode
Gain & black level	• Analog gain: 0 .. 18.06 dB (default 0 dB) • Digital gain: 0 .. 24.08 dB (default 0 dB) • Black level: 0 .. 2.55 DN (default 0 DN)
Auto functions	• Automatic black level
Signal/noise ratio	38 dB
Dynamic range	70 dB
Area of interest (AOI)	Horizontal and vertical AOI applicable directly at the sensor
Binning	• Horizontal: 2× 4× • Vertical: 2× 4×
Decimation	• Horizontal: 2× 3× 4× 5× 6× 7× • Vertical: 2× 3× 4× 5× 6× 7× 8×
Mount	C-mount (CS-mount on request)
Dimension	72×66×112 mm (W×H×L, without connectors)
Weight	620 g
Power input	10–26 VDC, 12 W (typical)
Digital i/o	4×opto in, 4×opto out, 2×TTL in, 2× TTL out, RS-232 interface
Supported operating systems	Microsoft Windows, Windows Embedded, Linux
Directly supported image processing libraries	Halcon, ActivVisionTools, Common Vision Blox, MATLAB, OpenCV
Ordering information	• 450801: CheckSight R502M/4GB • 450802: CheckSight R502M/4GB/WinXPe (Run time license of Windows Embedded Standard 2009 is preinstalled) Check latest price list or contact us to learn if more options are available.

Table A.26. CheckSight R502M technical specifications

Contacting Leutron Vision

Headquarters (Switzerland)

Address: Industriestrasse 57, CH-8152, Glattbrugg, Switzerland

Phone: ++41 44 809 88 22

Fax: ++41 44 809 88 29

E-mail (sales): <intsales@leutron.com>, e-mail (support): <intsupport@leutron.com>

Web: www.leutron.com[1]

Germany

Address: Macairestrasse 3, D-78467 Konstanz, Germany

Phone: ++49 7531 59 42 0

Fax: ++49 7531 59 42 99

E-mail (sales): <desales@leutron.com>, e-mail (support): <desupport@leutron.com>

Web: www.leutron.com/de/[2]

Other countries

Customers residing in other countries should have a look at the list of our representatives[3] for a distributor in their country. If no distributor exists in their country, they should contact the headquarter office directly.

Useful links

- To download software, documentation and other stuff, please visit our download area[4].
- Get more information about our support[5]. If you need to return some (defective) material to Leutron Vision, please visit the Return Material Authorization (RMA) page[6].
- Get information about prices and ordering[7].
- Find out detailed information about our hardware and software product range[8].

[1] http://www.leutron.com/
[2] http://www.leutron.com/de/
[3] http://www.leutron.com/sales-contact/
[4] http://www.leutron.com/support-downloads/download-area/
[5] http://www.leutron.com/support-downloads/
[6] http://www.leutron.com/rma/client_id.php
[7] http://www.leutron.com/price-request/
[8] http://www.leutron.com/products-machine-vision-image-acquisition/

Copyright © Leutron Vision